Міністерство освіти і науки України
НАЦІОНАЛЬНИЙ АВІАЦІЙНИЙ УНІВЕРСИТЕТ

Кафедра комп'ютеризованих систем управління

ДОПУСТИТИ ДО ЗАХИСТУ
Завідувач кафедри КСУ

_____ Олександр ЛИТВИНЕНКО

"_____"_____2023 р.

КВАЛІФІКАЦІЙНА РОБОТА
(ПОЯСНЮВАЛЬНА ЗАПИСКА)

ЗДОБУВАЧА ВИЩОЇ ОСВІТИ
ОСВІТНЬОГО СТУПЕНЯ «МАГІСТР»

Тема: Програмна система управління та резервного копіювання великих обсягів даних у хмарних середовищах

Виконавець: Олександр ГРАЧОВ

Керівник: Віталій НЕЧИПОРУК

Нормоконтролер: Євгеній ТУПОТА

Київ 2023

НАЦІОНАЛЬНИЙ АВІАЦІЙНИЙ УНІВЕРСИТЕТ

Факультет комп’ютерних наук та технологій

Кафедра комп’ютеризованих систем управління

Спеціальність 123 «Комп'ютерна інженерія

Освітньо-професійна програма «Системне програмування»

Форма навчання денна

ЗАТВЕРДЖУЮ

Завідувач кафедри КСУ

Олександр ЛИТВИНЕНКО

«_____»_____2023 р.

ЗАВДАННЯ

на виконання дипломної роботи (проекту)

Грачова Олександра Олександровича

1. Тема проекту (роботи): Програмна система управління та резервного копіювання великих обсягів даних у хмарних середовищах

затверджена наказом ректора від " 28 " серпня 2023 року № 1494/ст.

2. Термін виконання проекту (роботи): з 02.10.2023р. по 31.12.2023р.

3. Вихідні дані до проекту (роботи): Правила збирання та перетворення визначальних параметрів. Правила використання мов програмування. Стандарти ДСТУ. Положення про дипломні роботи.

4. Зміст пояснювальної записки: вступ, основи хмарних обчислень та управління даними, методи резервного копіювання великих обсягів даних, проектування та аналіз системи управління та резервного копіювання, розробка програмної системи управління та резервного копіювання., висновки

5. Перелік обов’язкового графічного матеріалу:

1) Діаграма варіантів використання;

2) Діаграма синхронізації бази даних із репозиторієм;

3) Діаграма побудови дерева каталогу документів репозиторію;

4) Діаграми трирівневого архітектурного рішення системи;

5) Скріншоти тестування програми.

6. Календарний план

№ п/п	Етапи виконання дипломної роботи	Термін виконання етапів	Примітка
1	Ознайомитись з постановкою задачі кваліфікаційної роботи	02.10 – 04.10	
2	Вивчити спеціальну літературу і технічну документацію	05.10 – 10.10	
3	Проведення аналізу існуючих засобів і методів вирішення поставленої задачі	11.10 – 25.10	
4	Проведення моделювання процесів системи (діаграми діяльності і послідовності)	26.10 – 02.11	
5	Проведення програмної реалізації системи	03.11– 17.11	
6	Тестування розроблених програм.	18.11 – 01.12	
7	Оформлення пояснювальної записки та обов’язкового графічного матеріалу	02.12 – 17.12	
8	Підготовка доповіді та презентації	18.12 – 22.12	

Дата отримання завдання: «02» 10. 2023 р.

Керівник кваліфікаційної роботи_____ Віталій НЕЧИПОРУК

Завдання прийняв до виконання _____ Олександр ГРАЧОВ

РЕФЕРАТ

Пояснювальна записка до дипломної роботи «Програмна система управління та резервного копіювання великих обсягів даних у хмарних середовищах»: 104 с., 57 рис., 5 табл., 25 літературних джерел та 2 додатки.

Ключові слова: СИСТЕМА УПРАВЛІННЯ ЕЛЕКТРОННИМИ ДОКУМЕНТАМИ, ХМАРНІ ОБЧИСЛЕННЯ, БАЗА ДАНИХ, ІНТЕГРАЦІЯ З ХМАРНИМИ СЕРВІСАМИ, ТЕСТУВАННЯ ПЗ, ВАЛІДАЦІЯ СИСТЕМИ.

Актуальність: Актуальність даної роботи полягає в необхідності розробки ефективних систем управління та захисту великих обсягів даних у хмарних середовищах, що впливає на сучасний технологічний ландшафт та сприяє досягненню цифрової безпеки і незалежності, зокрема в Україні.

Мета: розробка надійної та ефективної системи для зберігання та обробки файлів, інтегрованої з хмарними сховищами даних *Google Drive*, з використанням алгоритму шифрування даних *AES* для забезпечення високого рівня безпеки інформації.

Об’єкт дослідження: процес зберігання та обробки файлів у хмарних середовищах, зокрема використання хмарного сховища даних *Google Drive*.

Предмет дослідження: розробка системи управління та резервного копіювання файлів, яка інтегрована з хмарним сховищем даних *Google Drive*.

Методи дослідження: аналіз літератури та джерел, моделювання та проектування архітектури системи, експериментальна розробка та тестування.

Результати магістерської роботи рекомендується використовувати для розробки та впровадження корпоративних систем електронного документообігу, а також можуть бути корисними для наукових досліджень у галузі обробки і зберігання даних.

ЗМІСТ

ПЕРЕЛІК УМОВНИХ ПОЗНАЧЕНЬ, СИМВОЛІВ, СКОРОЧЕНЬ І ТЕРМІНІВ

ЕДС – електронна система документообігу, комп'ютеризована система для управління, зберігання та обміну електронними документами.

ІС – інформаційна система.

СУБД - система управління базами даних.

API – інтерфейс програмування додатків (*Application Programming Interface*), набір процедур, протоколів та інструментів для створення програмного забезпечення.

CRUD – створення, читання, оновлення, видалення (*Create, Read, Update, Delete*), чотири основні функції, які використовуються в програмах для взаємодії з базами даних.

JSON – *JavaScript Object Notation*, легкий формат обміну даними, легко читаний та пишеться людьми, легко аналізується та генерується машинами.

NuGet – менеджер пакетів для платформи *.NET*, який дозволяє розробникам легко додавати, оновлювати та видаляти бібліотеки та інструменти в їх проектах *.NET*.

SMTP – протокол простої передачі пошти (*Simple Mail Transfer Protocol*), використовується для відправки електронних листів.

SQL – мова структурованих запитів (*Structured Query Language*), використовується для проектування, управління та запитів до реляційних баз даних.

UI – користувацький інтерфейс (*User Interface*), засіб взаємодії між користувачем та програмним забезпеченням.

VS – *Visual Studio*, інтегроване середовище розробки від *Microsoft* для створення програм та веб-сервісів.

ВСТУП

У сучасному світі, де стрімкі технологічні зміни трансформують наше життя, важливість цифрових технологій неухильно зростає. Цифровізація проникла у всі сфери людської діяльності, від особистого використання до комплексних бізнес-процесів. В цьому контексті, здатність ефективно обробляти, зберігати та захищати великі обсяги даних стає критично важливою.

Ця тенденція не обмежується лише сферою інформаційних технологій. Вона впливає на наукові дослідження, де аналіз великих даних відіграє ключову роль у нових відкриттях, на фінансовий сектор, де точність та швидкість обробки фінансових транзакцій є необхідними для ефективного ринкового реагування, а також на багато інших галузей, від медицини до міського планування.

У цьому контексті, розробка надійних програмних систем управління та резервного копіювання даних, інтегрованих із хмарними середовищами, стає не тільки актуальною, але й необхідною задачею. Такі системи повинні вирішувати ключові виклики, пов'язані зі зберіганням великих обсягів даних: забезпечувати їх безпеку, здатність до масштабування, гнучкість доступу, а також високу ефективність обробки. Таким чином, розробка та впровадження ефективних систем управління даними у хмарних середовищах відіграє ключову роль у сучасному технологічному ландшафті, дозволяючи організаціям та індивідуальним користувачам оптимізувати свої процеси та забезпечити безпеку своїх даних.

Обсяги даних, що обробляються і зберігаються у хмарних середовищах, постійно зростають, зумовлюючи потребу в ефективних та надійних системах для їх управління та захисту. Це не лише технічна, а й стратегічна проблема, особливо для країн, які прагнуть забезпечити цифрову безпеку і незалежність, в тому числі і України. Використання хмарних технологій дає можливість ефективно масштабувати ресурси, але також ставить нові виклики перед розробниками програмного забезпечення у плані забезпечення безпеки та надійності.

Розвиток таких систем має велике значення не тільки для IT-сфери, але й для загальної економічної та соціальної інфраструктури країни. В Україні, де цифрова

трансформація набирає обертів, розробка ефективних та безпечних рішень для управління великими обсягами даних у хмарних середовищах може стати важливим кроком на шляху до технологічного лідерства та цифрової незалежності.

Виходячи з цього, дослідження, спрямоване на розробку таких програмних систем, є актуальним і важливим. Воно відповідає світовим тенденціям в галузі ІТ та вимогам часу, а також сприяє вирішенню конкретних задач, пов'язаних з обробкою та зберіганням великих обсягів даних, що є ключовими для розвитку сучасного інформаційного суспільства.

Об'єктом дослідження є процес зберігання та обробки файлів у хмарних середовищах, зокрема використання хмарного сховища даних *Google Drive*. Це включає аспекти пов'язані з резервним копіюванням, доступом до великих обсягів даних, їх шифруванням та захистом, а також ефективністю управління цими даними в хмарному середовищі.

Предметом дослідження є розробка системи управління та резервного копіювання файлів, яка інтегрована з хмарним сховищем даних *Google Drive* і використовує алгоритм шифрування *AES* для забезпечення безпеки даних.

Метою дослідження є розробка надійної та ефективної системи для зберігання та обробки файлів, інтегрованої з хмарними сховищами даних *Google Drive*, з використанням алгоритму шифрування даних *AES* для забезпечення високого рівня безпеки інформації.

Основні задачі дослідження:

– розробити архітектуру системи управління та резервного копіювання, що інтегрується з *Google Drive*, забезпечуючи ефективне зберігання та доступ до великих обсягів даних;

– реалізувати механізми шифрування даних за допомогою алгоритму *AES*, гарантуючи високий рівень безпеки інформації під час її зберігання та передачі;

– протестувати систему з погляду її надійності, швидкості обробки даних та здатності до масштабування в умовах збільшення обсягу даних;

– оцінити ефективність системи в реальних умовах експлуатації, включаючи аналіз безпеки, продуктивності та користувацького досвіду.

Для досягнення мети дослідження, яка полягає у розробці надійної та ефективної системи для зберігання та обробки файлів будуть застосовані наступні методи дослідження:

– аналіз літератури та джерел. Цей метод передбачає вивчення наявних теоретичних та практичних матеріалів у галузі хмарних обчислень, систем зберігання даних, методів шифрування та IT-безпеки. Аналіз літератури допоможе визначити найкращі практики, існуючі рішення, поточний стан технік та технологій, а також виявити можливі прогалини або вдосконалення;

– моделювання та проектування архітектури системи. Застосування методів системного аналізу та проектування для створення концептуальної та технічної архітектури запропонованої системи. Це включає моделювання робочих процесів, визначення ключових компонентів системи, їх взаємодії та способи інтеграції з хмарним сховищем *Google Drive*;

– експериментальна розробка та тестування. Реалізація прототипу системи з наступним її тестуванням. Це включає в себе розробку алгоритмів шифрування *AES*, інтеграцію з *Google Drive* та тестування системи на предмет її надійності, продуктивності, швидкості обробки даних, масштабування та безпеки.

Наукова новизна роботи полягає у :

– розробці унікальної архітектури, що оптимізує взаємодію між локальними системами та хмарними сховищами *Google Drive*, забезпечуючи високу продуктивність та ефективність управління даними;

– реалізації розширеного механізму шифрування з використанням *AES*. Адаптація та оптимізація алгоритму *AES* для забезпечення максимальної безпеки даних в хмарному середовищі, включаючи захист від зовнішніх загроз та внутрішніх вразливостей;

– розробці методів оптимізації продуктивності системи. Впровадження передових технологій та алгоритмів для забезпечення високої швидкості обробки та передачі великих обсягів даних, що є критично важливим для сучасних хмарних середовищ.

Практичне значення результатів магістерського дослідження, яке було зосереджене на створенні надійної та ефективної системи для зберігання та обробки файлів, інтегрованої з хмарними сховищами даних *Google Drive* має важливе практичне застосування:

– підвищення безпеки даних у корпоративному секторі. Розроблена система може бути застосована в різних організаціях для забезпечення високого рівня безпеки при зберіганні та обробці конфіденційних даних. Це особливо актуально для фінансових установ, медичних організацій та урядових агенцій, де захист інформації є критично важливим;

– оптимізація процесів зберігання та обробки даних. Система може бути використана для покращення ефективності обробки великих обсягів даних, зниження витрат на зберігання даних та оптимізації процесів доступу до них. Це стане в нагоді компаніям, що займаються аналітикою даних, хмарними сервісами та IT-інфраструктурою;

– вдосконалення хмарних сервісів. Результати дослідження можуть бути використані провідними постачальниками хмарних сервісів для розширення їх функціональності, зокрема, шляхом інтеграції покращеного шифрування та механізмів безпеки в їх продукти;

– застосування в академічних та наукових дослідженнях. Результати можуть бути корисними для наукових установ та університетів, які займаються дослідженнями в галузі кібербезпеки, хмарних обчислень та шифрування. Вони можуть використовувати ці результати для подальших досліджень та розробок;

– рекомендації щодо вдосконалення законодавства у сфері кібербезпеки. Результати можуть бути використані для розробки або удосконалення законодавчих та нормативних актів, що регулюють зберігання та обробку персональних даних, зокрема у хмарних середовищах.

РОЗДІЛ 1
ОСНОВИ ХМАРНИХ ОБЧИСЛЕНЬ ТА УПРАВЛІННЯ ДАНИМИ

1.1. Архітектура хмарних обчислень

Хмарні обчислення, часто згадувані просто як «хмара», це інноваційна технологія, яка дозволяє користувачам отримувати доступ до різноманітних обчислювальних ресурсів через Інтернет (рис. 1.1). Ці ресурси, що включають у себе сервери, мережі, сховища даних, програми та інші сервіси, можуть бути швидко налаштовані та звільнені з мінімальними зусиллями з боку користувача або потребою в управлінні. Основна ідея хмарних обчислень полягає в тому, що користувачі мають можливість доступу до своїх даних та додатків без необхідності управління або обслуговування базової інфраструктури [3].

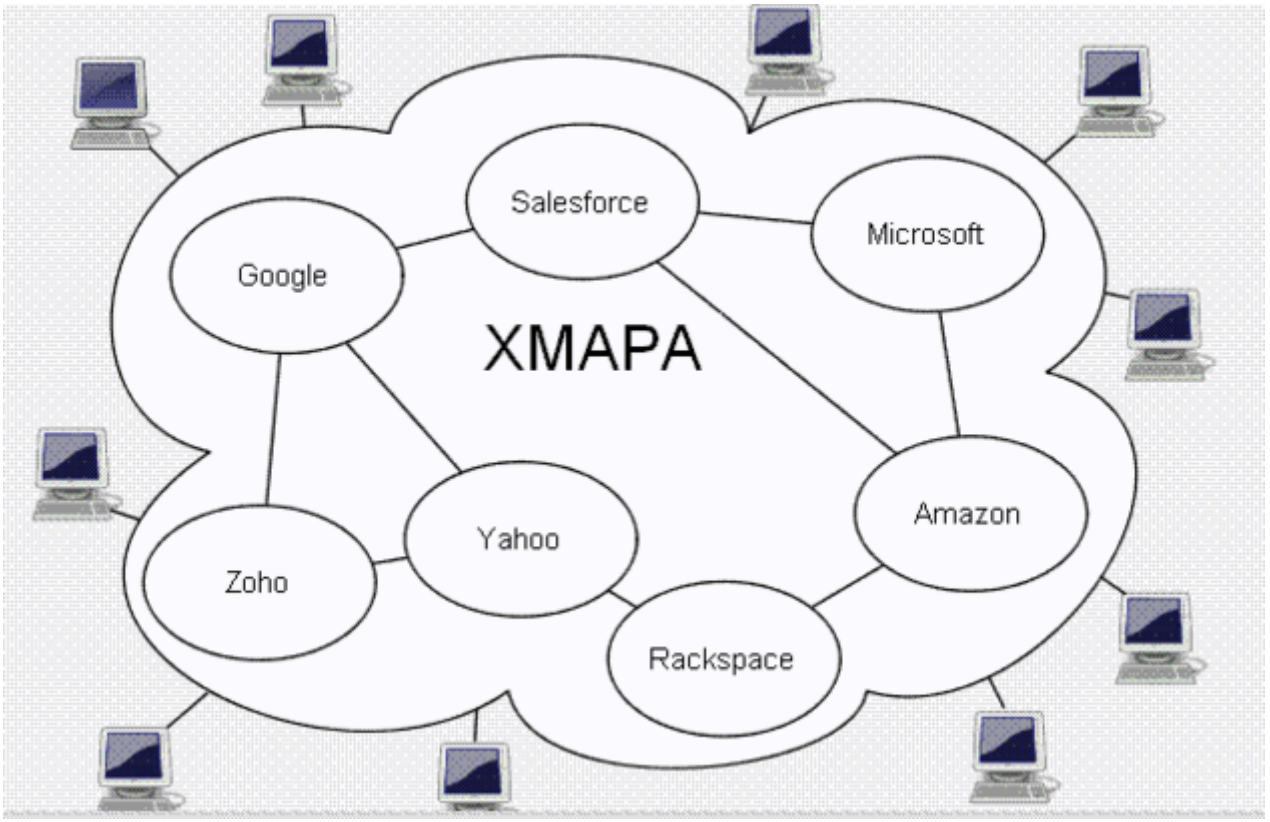

Рис. 1.1. Представлення хмарної архітектури у вигляді діаграми

У хмарних обчисленнях програмне забезпечення зазвичай надається як сервіс через Інтернет, тобто користувачі можуть використовувати програми та зберігати

дані в хмарі, не турбуючись про обслуговування апаратного та програмного забезпечення. Термін «хмара» використовується метафорично для опису Інтернету як абстракції, яка приховує технічні деталі. Як зазначається в документі *IEEE* 2008 року, хмарні обчислення представляють собою парадигму, в якій дані постійно зберігаються на серверах у мережі Інтернет і тимчасово кешуються на різних пристроях користувачів, таких як персональні комп'ютери, ігрові консолі, ноутбуки, смартфони та інші.

Термін «хмара» має своє коріння у сфері телефонії. До 1990-х років телекомунікаційні компанії переважно пропонували окремі лінії з'єднання, але з часом почали впроваджувати віртуальні приватні мережі (*VPN*), що надавали аналогічну якість обслуговування за значно нижчу вартість. Це стало можливим завдяки оптимізації використання мережевого трафіку. Символ хмари у цьому контексті використовувався для відокремлення мережі провайдера від кінцевих користувачів [4].

Важливий внесок у розвиток хмарних обчислень зробив *Amazon*, модернізувавши свої центри даних. Типово, більшість комп'ютерних мереж використовували лише близько 10% своєї потенційної потужності, щоб гарантувати надійність у разі раптового збільшення навантаження. Виявивши, що нова архітектура хмарних обчислень значно підвищує ефективність внутрішнього використання ресурсів, *Amazon* розпочав розробку продуктів, спрямованих на надання хмарних послуг зовнішнім клієнтам. Це призвело до запуску *Amazon Web Services* (*AWS*), заснованого на розподілених обчисленнях, у 2006 році [5].

У міру інтеграції хмарних технологій у бізнес-операції, управління їх складністю стає ключовим завданням. Компанії прагнуть краще управляти хмарними операціями, контролювати складність і використовувати спостережуваність для покращення ефективності операцій та оптимізації інвестицій у хмару [6].

Перехід до довгострокової гібридної моделі роботи спонукає попит на хмарні технології продуктивності та інструменти співпраці [7]. Хмарні інструменти для комунікацій, управління проектами, відеоконференцій та обміну файлами стають

невід'ємними для підтримки співпраці та продуктивності співробітників у гнучкому робочому середовищі.

Один із ключових трендів у сфері хмарних обчислень у 2023 році - це зосередження уваги на сталості та екологічності [8]. Компанії все частіше використовують хмарні технології не тільки для оптимізації бізнес-процесів, а й для досягнення своїх цілей у сфері сталого розвитку. Це означає перехід до більш стійких IT-інфраструктур, які зменшують вплив на навколишнє середовище, використовуючи хмарні ресурси більш ефективно та відповідально.

Хмарні технології дозволяють компаніям скоротити власні витрати на енергію та інші ресурси, так як управління даними та обчислювальні процеси переносяться у хмару, де провайдери хмарних послуг можуть масштабувати ресурси відповідно до потреб клієнтів. Це веде до оптимізації використання енергії та інших ресурсів, що в свою чергу сприяє зменшенню вуглецевого сліду компаній.

Використання хмарних технологій для сталого розвитку також включає в себе інтеграцію екологічних принципів і практик в діяльність компаній. Це може включати розробку IT-стратегій, які спрямовані на мінімізацію впливу на навколишнє середовище, наприклад, через використання обладнання, яке споживає менше енергії, або через використання відновлюваних джерел енергії.

Інтеграція штучного інтелекту (ШІ) в хмарні обчислення стає не просто модною тенденцією, але фундаментальною необхідністю для сучасних підприємств. ШІ використовується для автоматизації та оптимізації бізнес-процесів, підвищення точності аналізу даних та пришвидшення рішень, що підвищує оперативну ефективність і стимулює інноваційність.

Прийняття ШІ в хмарних обчисленнях також відкриває нові можливості для розширення бізнес-функціоналу. ШІ дозволяє компаніям аналізувати великі обсяги даних, автоматизувати складні процеси та підвищити якість обслуговування клієнтів. Багато організацій переносять свої робочі навантаження ШІ на публічні або гібридні хмари, щоб скористатися масштабованістю, гнучкістю та доступністю хмарних ресурсів. Це включає в себе використання хмарних платформ для розгортання ШІ-моделей, обробки великих даних та реалізації передових аналітичних рішень.

ШІ в хмарі також сприяє інноваціям у продуктовому дизайні, маркетингу, логістиці та інших аспектах діяльності компаній, відкриваючи нові шляхи для розвитку бізнесу. Використання алгоритмів машинного навчання та аналітики даних допомагає компаніям отримувати глибші розуміння процесів та прогнозувати майбутні тенденції, що є ключовим для конкурентоспроможності на ринку [9].

У сфері хмарних обчислень розрізняють кілька основних моделей надання послуг, кожна з яких пропонує унікальний набір функціональності та переваг для користувачів:

– програмне забезпечення як послуга (*SaaS*). Ця модель включає надання доступу до програмного забезпечення через інтернет, замість традиційного встановлення на локальні комп'ютери. Приклади *SaaS* включають такі популярні сервіси, як *Gmail* та *Google Docs*, які дозволяють користувачам використовувати поштовий сервіс та офісні додатки безпосередньо через веб-браузер, зберігаючи дані на віддалених серверах [10];

– платформа як послуга (*PaaS*). У цій моделі компанії надають розробникам інструменти та середовище для створення, тестування та розгортання додатків. Наприклад, *Google Apps* надає різноманітні додатки для бізнесу в онлайн-режимі, дозволяючи розробникам зосередитися на розробці додатків, в той час як управління інфраструктурою та платформою здійснюється *Google* [11];

– інфраструктура як послуга (*IaaS*). Ця модель пропонує базові обчислювальні ресурси, такі як віртуальні сервери, мережеве обладнання та сховище даних, які користувачі можуть використовувати на вимогу. Серед відомих постачальників *IaaS* - *Amazon*, *Microsoft*, *VMware*, *Rackspace* та *Red Hat*. Ці компанії пропонують широкий спектр послуг, від простого забезпечення базових обчислювальних ресурсів до складніших інтегрованих рішень [12].

Усі ці моделі надання послуг хмарних обчислень базуються на впевненості у тому, що інтернет здатний задовольнити потреби користувачів у обробці та зберіганні даних. Ця гнучкість та масштабованість хмарних послуг роблять їх привабливими для широкого кола користувачів, від індивідуальних розробників до великих корпорацій.

1.2. Моделі розгортання хмарних сервісів

Хмарні обчислення, які сьогодні є фундаментом для багатьох бізнес-процесів та інновацій, пропонують різноманітні моделі розгортання, щоб задовольнити унікальні потреби організацій. Вибір правильної моделі розгортання хмари залежить від багатьох факторів, включаючи вимоги до безпеки, масштабованість, вартість та специфічні бізнес-цілі. Моделі розгортання визначають, де і яким чином хмарні ресурси зберігаються, керуються та доступні для користувачів.

Моделі розгортання хмарних обчислень включають:

Публічні хмари

Публічні хмарні сервіси часто пропонують безкоштовні періоди для тестування, дозволяючи користувачам оцінити їх функціональність та ефективність перед переходом на платну модель. Цей підхід має на меті забезпечити користувачам високоякісні послуги та оптимальну інфраструктуру. Публічні хмари, як правило, використовуються приватними особами та компаніями, які шукають надійне та безпечне середовище для своїх даних і додатків, подібне до того, яке пропонують приватні хмари (рис. 1.2).

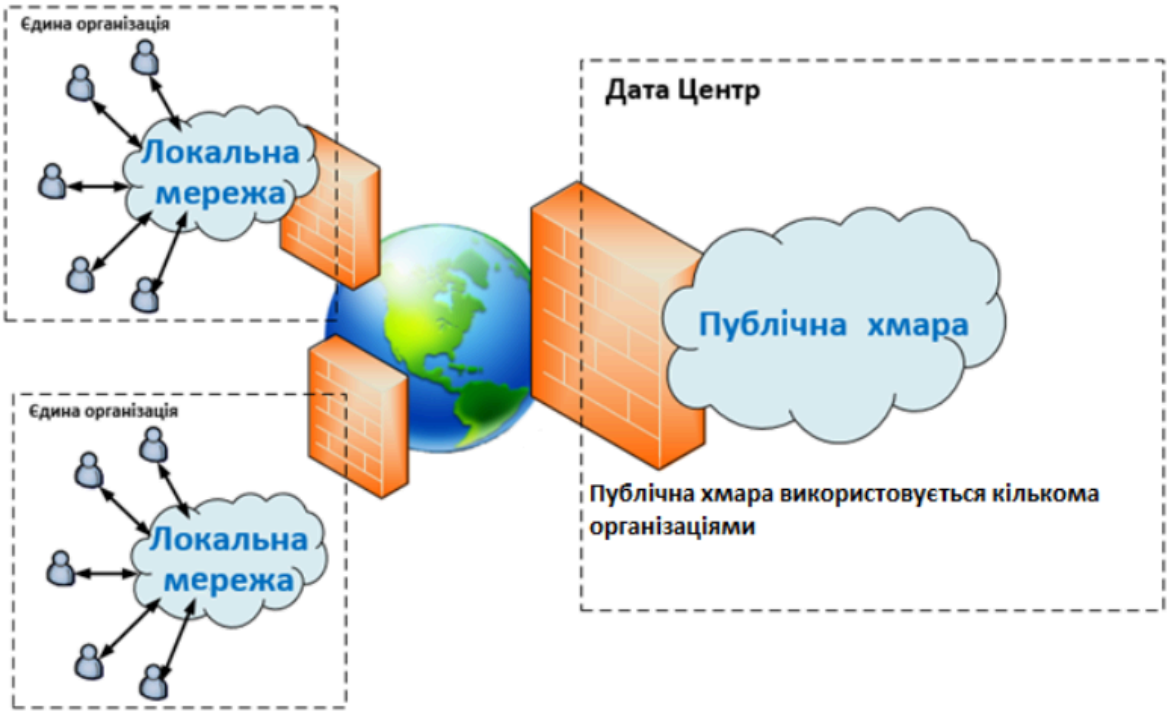

Рис. 1.2. Модель публічної хмари

Відмінність публічної хмари від приватної полягає у тому, що ресурси публічної хмари доступні для широкого кола користувачів, а не обмежені однією організацією. У публічних хмарах інфраструктура та обчислювальні ресурси знаходяться у віддалених центрах обробки даних і управляються хмарним провайдером. Це дозволяє компаніям оптимізувати свої витрати та збільшити ефективність операцій, використовуючи масштабованість та гнучкість хмарних рішень.

Крім того, публічні хмари забезпечують високий рівень доступності та надійності, що робить їх привабливим вибором для бізнесів, які потребують стабільного та безпечного онлайн-простору для своєї діяльності.

Архітектура публічної хмари розроблена так, що дозволяє багатьом користувачам спільно використовувати комп'ютерні ресурси. Це досягається завдяки розділенню даних між різними клієнтами (орендарями) таким чином, що інформація кожного з них залишається ізольованою та захищеною від доступу інших. Ефективність публічної хмари забезпечується швидким обміном даними та залежить від якісного мережевого з'єднання з високою пропускною спроможністю.

Економічна вигода використання публічної хмари полягає у можливості залучення значних обчислювальних ресурсів за оптимальну вартість. Це робить публічну хмару привабливою з точки зору економії, оскільки користувачі сплачують лише невелику плату за використання, що робить її варіантом з низькими оперативними витратами. Таким чином, публічна хмара стає доступною та вигідною для широкого кола користувачів, від індивідуальних осіб до великих організацій.

Переваги публічної хмари:

– економічна ефективність. Публічні хмари часто виявляються більш вигідними з фінансової точки зору, оскільки вони пропонують модель оплати за використання. Користувачі сплачують лише за ті ресурси, які вони використовують, знижуючи загальні витрати на IT;

– гнучкість та масштабованість. Публічні хмари надають величезні можливості для масштабування, дозволяючи користувачам легко збільшувати або зменшувати використані ресурси в залежності від поточних потреб;

– безперебійна робота та надійність. Більшість публічних хмарних провайдерів гарантують високий рівень доступності та надійності своїх послуг, забезпечуючи безперебійну роботу критично важливих додатків та сервісів;

– широкий спектр послуг. Публічні хмари пропонують різноманітність сервісів та рішень, від базових обчислювальних ресурсів до складних інтегрованих платформ, які задовольняють різні потреби користувачів.

Недоліки публічної хмари:

– проблеми з безпекою даних. Оскільки дані зберігаються на віддалених серверах, існує ризик несанкціонованого доступу та порушення конфіденційності інформації, особливо для організацій, які мають суворі вимоги до безпеки;

– обмеження контролю. У випадку публічних хмар, організації мають обмежений контроль над інфраструктурою та платформами, що може ускладнити інтеграцію з існуючими системами або дотримання конкретних вимог;

– залежність від Інтернету: Публічні хмари залежать від інтернет-з'єднання, що може створювати проблеми у випадку низької пропускної спроможності мережі або перебоїв у її роботі;

– стандартизовані послуги. Публічні хмари часто пропонують стандартизовані послуги, які можуть не відповідати специфічним вимогам або варіантам використання певних організацій, обмежуючи можливості їх використання.

Отже, публічні хмарні сервіси пропонують значні переваги, такі як економічна ефективність, висока гнучкість, надійність та широкий спектр послуг. Вони є вигідним рішенням для організацій і приватних осіб, які шукають масштабованих та доступних технологічних рішень. Однак, публічні хмари також мають певні обмеження, включаючи проблеми з безпекою даних, обмежений контроль над інфраструктурою, залежність від інтернет-з'єднання та стандартизованість послуг, що може не відповідати всім специфічним потребам. В цілому, публічні хмари є

цінним інструментом у сучасному технологічному ландшафті, проте їх використання вимагає ретельного розгляду потреб і ризиків [13].

Гібридні хмари

Гібридні хмарні обчислення комбінують елементи як приватних, так і публічних хмарних платформ (рис. 1.3). Це дає змогу розподіляти навантаження між різними середовищами, забезпечуючи вищу гнучкість у використанні ресурсів. Такий підхід дозволяє компаніям адаптуватися до змін у вимогах до обчислювальних ресурсів, ефективно вирішуючи потенційні проблеми з переповненням інфраструктури.

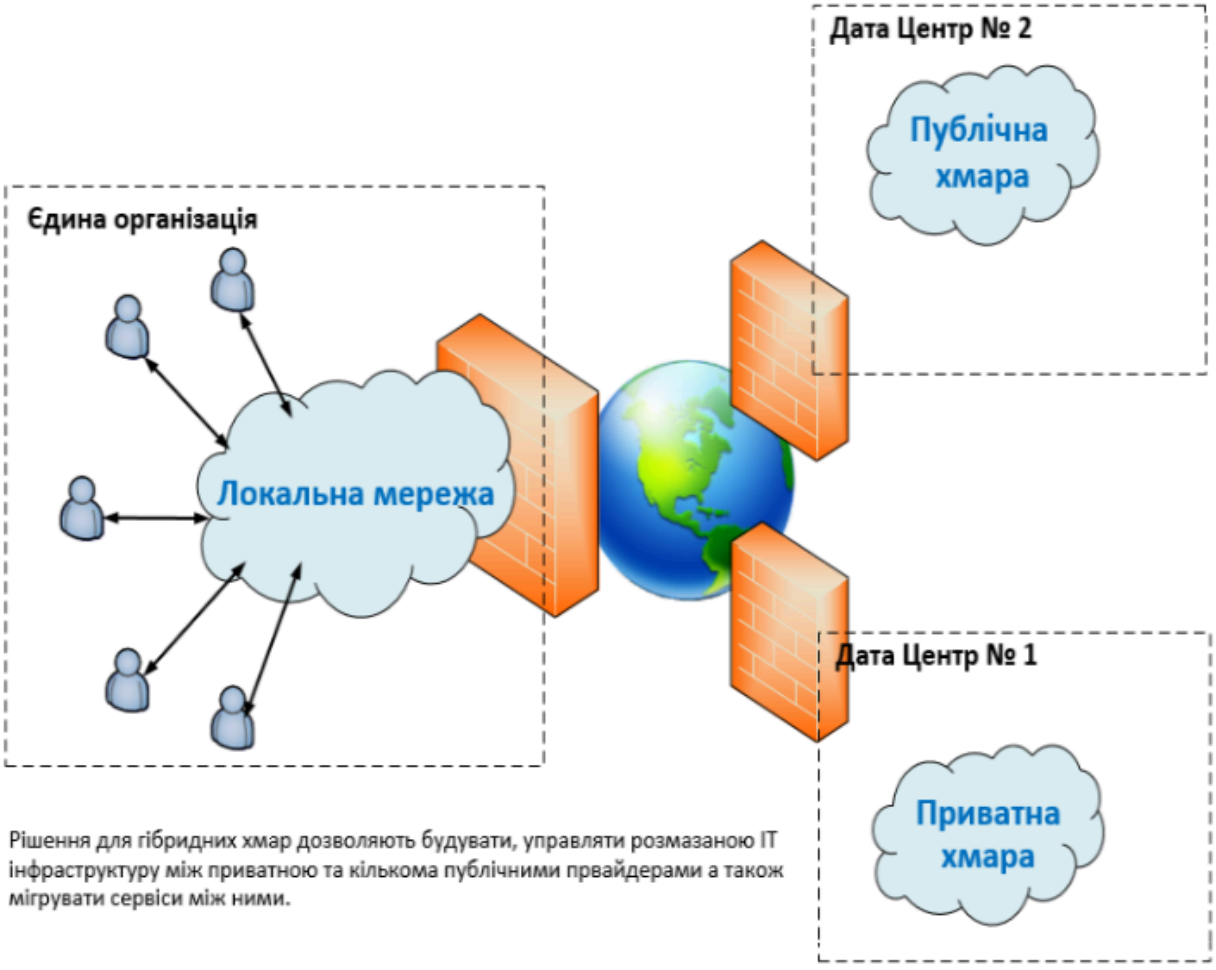

Рис. 1.3. Модель гібридної хмари

З іншого боку, гібридні хмари мають певні обмеження, включаючи обмежений прямий доступ до повного контролю над всіма аспектами їх архітектури. Компанії, які використовують гібридні хмарні рішення, можуть зіткнутися з викликами, пов'язаними з сумісністю та інтеграцією різних систем, включаючи локальні мережі

та пристрої зберігання даних. Це вимагає ретельного планування та впровадження додаткових ресурсів для забезпечення ефективної координації між різними хмарними середовищами.

Архітектура гібридних хмарних обчислень складається з трьох ключових компонентів, кожен з яких відіграє важливу роль у забезпеченні ефективного та безпечного обміну даними між різними хмарними середовищами:

– публічна інфраструктура як послуга (*IaaS*). Цей компонент надає базові обчислювальні ресурси, такі як віртуальні сервери, сховища даних та мережеві компоненти, доступні через інтернет. Також він забезпечує масштабованість та гнучкість, дозволяючи користувачам швидко збільшувати або зменшувати ресурси в залежності від потреб;

– приватна хмара. Це контрольоване, захищене середовище, яке використовується виключно однією організацією. Приватна хмара може бути розміщена локально в центрі даних організації або управлятися третьою стороною. Вона забезпечує більший контроль та безпеку, що є особливо важливим для даних, які вимагають суворого дотримання норм і стандартів конфіденційності та безпеки;

– мережа широкої області (*WAN*). Це мережевий компонент, який забезпечує з'єднання між публічною та приватною хмарами. *WAN* дозволяє безпечно передавати дані між різними середовищами, забезпечуючи необхідну пропускну спроможність та низьку затримку. Це особливо важливо для сценаріїв, де потрібно швидко переміщувати великі обсяги даних або підтримувати стабільність взаємодії між різними обчислювальними середовищами.

Ці компоненти разом формують гібридну хмару, яка поєднує гнучкість та масштабованість публічної хмари з контролем та безпекою приватної хмари, надаючи організаціям більш варіативні та ефективні можливості для управління їхніми IT-ресурсами.

Переваги гібридних хмарних сервісів:

– гнучкість та масштабованість. Гібридні хмари дозволяють компаніям масштабувати ресурси вгору або вниз відповідно до потреб, комбінуючи найкращі особливості приватних та публічних хмар. Це дозволяє користувачам гнучко

використовувати обчислювальні ресурси, оптимізуючи витрати та реагуючи на зміни у бізнес-вимогах;

– оптимальне використання ресурсів. Гібридні хмари дозволяють компаніям зберігати чутливі або критично важливі дані у приватній хмарі, в той час як менш чутливі дані та додатки можуть використовувати ресурси публічної хмари, забезпечуючи більш ефективне використання ресурсів;

– безпека та відповідність вимогам. Гібридні хмари дозволяють компаніям використовувати строгі міри безпеки приватних хмар для критично важливих даних та операцій, одночасно використовуючи гнучкість публічних хмар для менш чутливих задач.

Недоліки гібридних хмарних сервісів:

– складність управління. Гібридні хмарні сервіси можуть бути складнішими в управлінні через необхідність координації між різними середовищами хмар (приватними та публічними) та вирішення вимог до сумісності;

– вища вартість. Через їх складність та необхідність інтеграції різних систем, гібридні хмари можуть вимагати більших витрат на розробку та підтримку порівняно з одної хмарною моделлю;

– проблеми з безпекою та приватністю. Хоча гібридні хмари дозволяють застосовувати розширені міри безпеки, інтеграція різних хмарних середовищ також може створювати нові виклики для безпеки та приватності, особливо при переміщенні даних між хмарами.

Отже, гібридні хмарні сервіси поєднують переваги приватних та публічних хмар, надаючи гнучкість, масштабованість та оптимізоване використання ресурсів. Вони дозволяють компаніям зберігати чутливі дані у приватних хмарах для забезпечення безпеки, в той час як менш чутливі додатки можуть використовувати ресурси публічної хмари для більшої ефективності. Однак, гібридні хмари також мають свої недоліки, включаючи складності управління, потенційно вищі витрати та виклики з безпекою та приватністю. Незважаючи на це, вони залишаються цінним варіантом для організацій, які шукають збалансоване рішення, що забезпечує гнучкість хмарних обчислень разом із контролем та безпекою [14].

Приватна хмара

Модель приватної хмари пропонує високий рівень безпеки та контролю, оскільки її використання обмежується однією компанією або організацією. Ця модель хмари забезпечує гнучкість управління та можливість керувати даними та ресурсами в умовах підвищеної конфіденційності. Приватна хмара базується на пулі обчислювальних ресурсів, які забезпечують потужність обчислень у віртуалізованому середовищі (рис. 1.4).

Рис. 1.4. Модель приватної хмари

Цей тип хмари використовується в рамках моделі «інфраструктура як послуга» (*IaaS*), де постачальник основної інфраструктури може відрізнятися від постачальника послуг віртуалізації на цій інфраструктурі. Приватна хмара надає організаціям перевагу повного контролю над своїми обчислювальними середовищами, що є важливим для забезпечення суворих вимог до безпеки та дотримання корпоративних політик і стандартів.

Переваги приватних хмарних сервісів:

– підвищена безпека та конфіденційність. Приватні хмари забезпечують більший контроль над інфраструктурою та даними, що важливо для організацій з високими вимогами до безпеки та конфіденційності. Це дозволяє компаніям краще захищати свої чутливі дані та дотримуватися нормативних стандартів;

– налаштування та контроль. Оскільки приватні хмари належать та експлуатуються однією організацією, вони забезпечують більшу свободу у

налаштуванні та керуванні обчислювальними ресурсами, відповідно до специфічних потреб і політик організації;

– зменшення зовнішньої залежності. Використання приватних хмар може зменшити залежність від зовнішніх провайдерів хмарних послуг, надаючи організаціям більшу незалежність та контроль над своїми IT-ресурсами.

Недоліки приватних хмарних сервісів:

– вища вартість. Побудова та управління приватною хмарою часто вимагають значних капіталовкладень та операційних витрат, що робить її дорожчою в порівнянні з публічними хмарними рішеннями;

– складність управління та технічного обслуговування. Приватні хмари вимагають складного управління та обслуговування, включаючи необхідність мати кваліфікований IT-персонал для забезпечення безперебійної роботи та підтримки інфраструктури;

– обмеження масштабованості. У порівнянні з публічними та гібридними хмарами, приватні хмари можуть мати обмеження у плані швидкого масштабування ресурсів, що може бути важливим фактором для організацій, які потребують гнучкості та здатності швидко реагувати на зміни в бізнес-потребах.

Отже, приватні хмарні сервіси пропонують високий рівень безпеки, конфіденційності та контролю над ресурсами, що робить їх ідеальним рішенням для організацій з суворими вимогами до даних та власною IT-інфраструктурою. Вони забезпечують більшу свободу у налаштуванні та оптимізації, але це може призвести до вищих витрат на реалізацію та управління. Незважаючи на свою складність і потенційно вищі витрати, приватні хмари залишаються важливим варіантом для компаній, які прагнуть до максимального контролю та безпеки своїх даних [15].

1.3. Основні підходи до управління даними в хмарних середовищах

Управління даними в хмарних середовищах – це комплексний процес, який включає збір, зберігання, обробку та аналіз даних у хмарних інфраструктурах. Він вимагає ретельного підходу до безпеки, конфіденційності, доступності та інтеграції

даних. Ефективне управління даними у хмарі дозволяє організаціям підвищити продуктивність, оптимізувати витрати та гнучко реагувати на зміни бізнес-вимог. Це включає в себе визначення того, як довго і де зберігати різні типи даних, а також їх безпеку та шифрування. Основні аспекти, на які необхідно звернути увагу при управлінні даними в хмарних середовищах:

Модель зберігання даних

Управління даними в хмарних середовищах вимагає вибору ефективної моделі зберігання, яка відповідає конкретним потребам та вимогам бізнесу. Вибір правильної моделі забезпечує не тільки оптимальне зберігання даних, але й сприяє підвищенню ефективності, доступності та безпеки інформації. Важливо враховувати не тільки поточні потреби бізнесу, але й можливість майбутнього розширення та змін у використанні даних. Ефективне управління даними в хмарі включає розгляд таких ключових моделей зберігання:

– об'єктне зберігання. Ця модель ідеально підходить для зберігання великих обсягів неструктурованих даних, таких як медіафайли, фотографії, та інші великі документи. Завдяки своїй структурі, об'єктне зберігання дозволяє легко масштабувати та управляти великими наборами даних;

– блокове зберігання. Цей тип зберігання використовується для сценаріїв, які вимагають високої продуктивності, таких як бази даних та операційні системи. Він забезпечує швидкий доступ до даних і може бути ефективно інтегрований з різними програмами та системами;

– файлове зберігання. Ця модель використовується для зберігання файлів у вигляді ієрархічної файлової системи, що робить її зручною для спільного доступу та співпраці. Файлове зберігання дозволяє користувачам легко організовувати, управляти та спільно використовувати файли.

Кожна з цих моделей має свої унікальні переваги і може бути адаптована до різних бізнес-сценаріїв, забезпечуючи гнучке та ефективне управління даними в хмарних середовищах [16].

Управління життєвим циклом даних

Управління життєвим циклом даних (*Data Lifecycle Management, DLM*) призначене для забезпечення ефективності, безпеки та оптимального використання ресурсів. Цей процес охоплює ряд дій, спрямованих на управління даними від моменту їх створення до кінцевого видалення або архівації.

Починається управління життєвим циклом даних з визначення політики зберігання. Це включає рішення про те, як довго і де будуть зберігатися різні типи даних. Наприклад, деякі дані можуть вимагати короткотермінового зберігання через їх оперативний характер, тоді як інші – довгострокового зберігання з міркувань відповідності нормативним вимогам або історичної цінності.

Наступним йде процес архівації та видалення. Старі або рідко використовувані дані часто переносяться в архів для звільнення обчислювальних ресурсів та зниження витрат. З іншого боку, застарілі або непотрібні дані видаляються для забезпечення безпеки та ефективності системи.

Також важливим аспектом управління життєвим циклом даних є періодичні перевірки для забезпечення актуальності даних. Це включає перегляд політик зберігання, аналіз потреб користувачів та бізнес-процесів, а також впровадження відповідних змін для забезпечення того, що системи зберігання даних залишаються оптимальними та відповідають змінюваним вимогам бізнесу.

В цілому, управління життєвим циклом даних в хмарних середовищах вимагає постійного моніторингу, оцінки та адаптації до змінюваних потреб бізнесу та технологій, гарантуючи, що дані завжди захищені, доступні та використовуються ефективно [17].

Безпека даних та шифрування

В умовах зростаючої кіберзагрози та потреби у захисті конфіденційної інформації, важливо впроваджувати ефективні заходи безпеки. Однією з ключових складових безпеки даних є шифрування. Шифрування даних у стані спокою (коли дані зберігаються на серверах) запобігає несанкціонованому доступу до інформації у випадку витоку даних або порушення системи зберігання. Шифрування даних під час передачі (коли дані передаються між різними вузлами або з сервера на клієнтські

пристрої та навпаки) гарантує, що інформація залишається захищеною від перехоплення або модифікації третіми сторонами.

Ще одним критичним аспектом безпеки даних є аутентифікаційні механізми та контроль доступу. Сильні процедури аутентифікації, такі як багатофакторна аутентифікація, гарантують, що доступ до даних мають лише уповноважені особи. Контроль доступу, який включає в себе розмежування прав і обмеження доступу до конкретних наборів даних, запобігає несанкціонованому використанню або зміні інформації.

Комплексний підхід до безпеки даних у хмарі, який включає шифрування, аутентифікацію та контроль доступу, дозволяє організаціям захищати свої важливі дані від зовнішніх загроз та внутрішніх витоків інформації. Такий підхід вимагає ретельного планування та впровадження, але є невід'ємною частиною стратегії цифрової безпеки сучасного бізнесу.

Резервне копіювання та відновлення

Реалізація надійної стратегії резервного копіювання та відновлення даних забезпечує запобігання втратам важливої інформації та гарантує здатність бізнесу швидко відновитися після непередбачуваних подій, таких як технічні збої, кібератаки чи природні катастрофи. Основою стратегії резервного копіювання є створення і зберігання копій усіх критично важливих даних. Це означає регулярне копіювання інформації та її зберігання в безпечному, відокремленому місці. Важливо, щоб резервні копії були розподілені, тобто зберігалися в іншому місці, ніж основні дані, щоб забезпечити їх збереження в разі локальних проблем.

Додатковим критичним компонентом стратегії є тестування та перевірка процесів відновлення даних. Це включає періодичне тестування резервних копій, щоб переконатися, що вони можуть бути ефективно використані для відновлення даних у випадку необхідності. Тестування допомагає виявити та виправити будь-які проблеми з резервними копіями до того, як вони стануть критичними.

Крім того, важливо мати чіткий план дій для відновлення даних у випадку їх втрати, що включає процедури, кроки та відповідальних осіб. Цей план повинен

бути документований та регулярно оновлюватися для врахування будь-яких змін у хмарній інфраструктурі або бізнес-процесах.

Належне резервне копіювання та відновлення даних є важливими для забезпечення стійкості бізнесу, зменшення ризику втрати інформації та гарантування безперебійності операцій.

Контроль якості даних

В середовищах, де дані постійно змінюються та оновлюються, підтримка високого рівня якості даних стає особливо важливою. Один з основних елементів контролю якості даних - це процес очищення даних. Це включає виправлення помилок, видалення дублікатів та вирішення несумісності. Очищення даних допомагає забезпечити, що інформація точна та відповідає встановленим стандартам.

Валідація даних також є важливою частиною процесу контролю якості. Вона полягає у перевірці, чи відповідають дані певним критеріям або форматам, наприклад, чи вірні формати дат, адрес, номерів телефонів та інших критичних даних. Це важливо для забезпечення консистентності та достовірності даних у хмарних системах. Також сюди варто віднести і управління метаданими. Ефективне управління метаданими дозволяє краще розуміти контекст і використання даних, а також сприяє легшому пошуку та класифікації інформації.

В цілому, контроль якості даних у хмарних середовищах вимагає систематичного підходу та постійного моніторингу, щоб забезпечити, що дані залишаються точними, актуальними та корисними для бізнесу. Це допомагає знизити ризики, пов'язані з неправильними даними, та підвищує ефективність прийняття рішень.

Сумісність і інтеграція

Важливість цього аспекту полягає в тому, щоб гарантувати, що нові хмарні рішення можуть працювати безперебійно з існуючою інфраструктурою, включаючи інші хмарні платформи, локальні системи, бази даних та бізнес-додатки. Забезпечення сумісності передбачає, що хмарні рішення будуть відповідати стандартам та протоколам, які вже використовуються в існуючих системах. Це

означає, що дані можуть бути легко передані та оброблені між різними системами без необхідності складної та трудомісткої адаптації або перетворення.

Інтеграція хмарних рішень також має велике значення. Це означає, що нові хмарні сервіси можуть бути ефективно інтегровані в існуючу IT-архітектуру, забезпечуючи плавне та ефективне функціонування бізнес-процесів. Інтеграція може охоплювати різні аспекти, від взаємодії між різними додатками та базами даних до забезпечення єдиної точки доступу до даних через різні платформи.

Ефективна сумісність та інтеграція вимагають ретельного планування, оцінки технологічних можливостей і потреб, а також чіткого розуміння поточної IT-інфраструктури. Це включає вибір рішень, які можуть легко інтегруватися, та розробку стратегій, які дозволять гладко впровадити нові технології без збоїв у бізнес-операціях. Такий підхід дозволяє організаціям максимально використовувати переваги хмарних технологій, одночасно забезпечуючи стабільність та ефективність існуючих систем [18].

1.4. Проблеми безпеки та конфіденційності в хмарних сервісах

Коли користувачі переходять до використання хмарних сервісів, вони часто стикаються з питаннями, пов'язаними з переміщенням своїх конфіденційних даних та додатків з локальних обчислювальних середовищ до загальнодоступних хмарних платформ. Одним з основних занепокоєнь є ризик несанкціонованого доступу до інтерфейсів управління хмарними сервісами. В традиційних центрах обробки даних, інтерфейси зазвичай доступні лише авторизованим адміністраторам і часто через захищені приватні мережі. Натомість, у хмарних сервісах, інтерфейси управління часто доступні онлайн через веб-додатки, що потенційно підвищує ризики безпеки, оскільки такі системи можуть бути доступні не тільки авторизованим користувачам, але й потенційним зловмисникам.

Крім того, оскільки доступ до управління хмарними ресурсами здійснюється зазвичай через веб-інтерфейси та/або сервісні технології, інтерфейси управління хмарами можуть успадковувати вразливості цих технологій (рис. 1.5). Це означає,

що будь-які слабкі місця у безпеці веб-додатків або сервісних технологій можуть бути експлуатовані для несанкціонованого доступу або атак на хмарні системи.

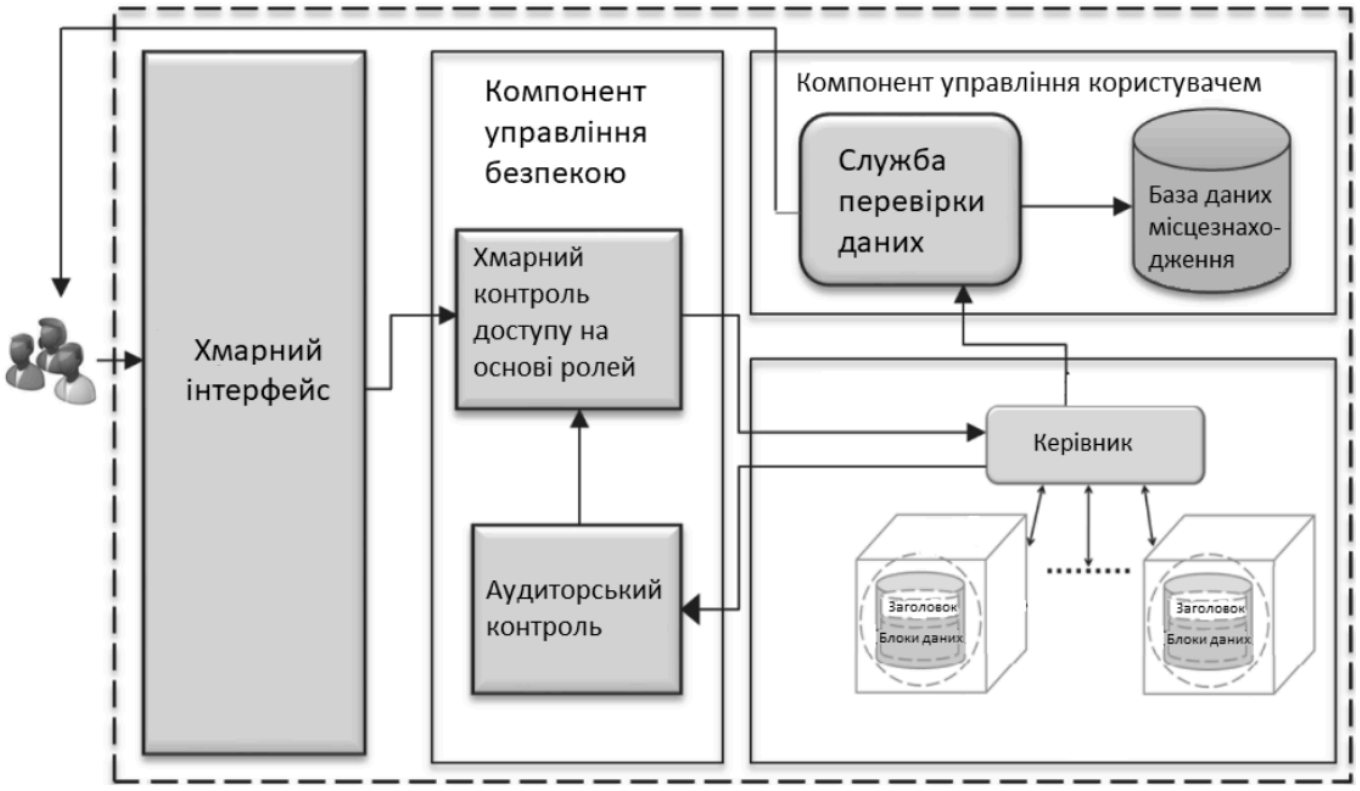

Рис. 1.5. Реалізація доступу до хмарних сервісів

В цілому, ці проблеми вимагають ретельного підходу до безпеки в хмарних сервісах, включаючи сильні аутентифікаційні механізми, шифрування даних, регулярне оновлення та вдосконалення інтерфейсів управління для запобігання вразливостям.

В хмарних сервісах існує кілька ключових проблем безпеки та конфіденційності. По-перше, є ризик втрати даних, якщо постачальник хмарних послуг не забезпечує належних заходів резервного копіювання. По-друге, крадіжка облікових даних у хмарному середовищі може дозволити зловмисникам перехоплювати або підробляти дані. Третьою проблемою є незахищені інтерфейси та *API*, що піддають організацію ризику різних загроз. *DDoS*-атаки можуть перевантажити інфраструктуру хмарних сервісів, блокуючи доступ користувачів. Інсайдери з непорядними намірами в *IaaS*, *PaaS*, або *SaaS* середовищах можуть отримати доступ до конфіденційної інформації. Також, хмарні обчислення можуть бути використані зловмисниками для виконання потужних хакерських атак [19].

1.5. Висновки до розділу

У рамках даного розділу було розглянуто основні питання, пов'язані з архітектурою хмарних обчислень, їх моделями розгортання, а також з підходами до управління даними в хмарних середовищах. Особливу увагу приділено аналізу хмарних технологій з точки зору їх впливу на сталість та екологічність, а також інтеграції штучного інтелекту в хмарні обчислення. Розглянуто основні моделі надання послуг у хмарних обчисленнях: програмне забезпечення як послуга, платформа як послуга, та інфраструктура як послуга. Виділено основні моделі розгортання хмарних сервісів, такі як: публічні хмари, гібридні хмари та приватні хмари. Висвітлено ключові аспекти управління даними, включаючи зберігання даних, управління життєвим циклом даних, безпеку, шифрування, резервне копіювання та відновлення. Також, розглянуто проблеми безпеки та конфіденційності, з якими користувачі можуть зіткнутися під час використання хмарних сервісів.

РОЗДІЛ 2
МЕТОДИ РЕЗЕРВНОГО КОПІЮВАННЯ ВЕЛИКИХ ОБСЯГІВ ДАНИХ

2.1. Традиційні та сучасні стратегії резервного копіювання

Частота створення повних резервних копій може істотно відрізнятися, адже вона залежить від темпів змін у даних та важливості уникнення втрати цих даних. Однак є загальноприйнятою практикою виконувати повне резервне копіювання під час вихідних або поза робочим часом, щоб мінімізувати кількість змін, які відбуваються під час процесу копіювання.

Перед тим як диски стали основним засобом резервного копіювання на початку 2000-х, багато організацій використовували стрічкові бібліотеки для зберігання резервних копій даних. Хоча магнітні стрічки досі використовуються, особливо для архівних даних, які не вимагають швидкого доступу, деякі компанії перейшли до знімних жорстких дисків. Однак загальна концепція зберігання даних на знімному носії залишилася сталою.

Перехід до дискового резервного копіювання дозволив організаціям забезпечити більш надійний захист даних. Зазвичай, перед введенням дискового резервного копіювання, організації виконували лише одне нічне резервне копіювання. Спочатку ці нічні резервні копії були повними, але з ростом обсягів даних та зменшенням часового вікна для резервного копіювання багато організацій перейшли до виконання інкрементних нічних резервних копій.

На початку розвитку дискового резервного копіювання, програми для цих цілей розроблялися з урахуванням роботи на окремому сервері. Таке програмне забезпечення координувало процес копіювання і записувало резервні дані на спеціалізовані масиви зберігання. Швидко набувши популярності, ці системи вирізнялися здатністю здійснювати резервне копіювання та відновлення даних у мережі, уникаючи потреби у фізичному використанні стрічок.

Хоча деякі програми резервного копіювання досі базуються на окремих серверах, все більше розробників орієнтується на впровадження інтегрованих рішень для захисту даних. Простіші інтегровані системи передачі даних, по суті, є файловими серверами з вмонтованими жорсткими дисками та відповідним програмним забезпеченням. Такі *plug-and-play* пристрої зберігання часто оснащені автоматизованими функціями для моніторингу використання диска, розширення сховища та інтегрованих стрічкових бібліотек.

Також, деякі виробники пропонують резервні системи, що базуються на гіперконвергентних технологіях. Такі системи складаються з набору уніфікованих серверів, що об'єднані для спільного виконання процесів резервного копіювання. Основна перевага гіперконвергентних систем полягає в їх масштабованості: кожен вузол містить власні сховища, обчислювальні та мережеві ресурси, дозволяючи адміністраторам масштабувати потужність організації шляхом додавання нових вузлів.

Більшість сучасних дискових резервних пристроїв, незалежно від їх конструкції, дозволяють переміщати дані з обертових носіїв на магнітні стрічки для довгострокового зберігання. Стрічкові системи залишаються важливими через підвищення щільності стрічки та розвиток лінійних файлових систем для стрічок.

У ранній фазі розвитку дискових резервних систем, такі системи часто називали віртуальними стрічковими бібліотеками (*VTL*), оскільки вони імітували функціональність стрічкових накопичувачів. Це означало, що існуюче програмне забезпечення для резервного копіювання на стрічки могло без проблем працювати з цими дисковими системами, сприймаючи їх як фізичні стрічкові бібліотеки. Проте з часом, коли розробники програмного забезпечення адаптували свої продукти для оптимальної роботи з дисками, популярність віртуальних стрічкових бібліотек знизилася.

Твердотільні накопичувачі (*SSD*) рідко використовуються в якості основного рішення для резервного копіювання, головним чином через їх високу вартість і обмежену витривалість. Втім, деякі системи зберігання використовують *SSD* як частину кешування для оптимізації процесу запису даних на дискові масиви,

особливо це характерно для гіперконвергентних систем. Дані спочатку кешуються у флеш-пам'яті, а потім переносяться на жорсткий диск. Зі зростанням ємності *SSD*, ці накопичувачі можуть знайти застосування у сфері резервного копіювання.

Сучасні системи первинного зберігання даних еволюціонували, пропонуючи розширені можливості для резервного копіювання, включаючи розширені схеми *RAID*, необмежене створення знімків, а також інструменти для реплікації цих знімків у додаткові резервні копії. Однак, не дивлячись на ці інновації, резервне копіювання на основі первинних сховищ зазвичай є дорожчим і не володіє такими можливостями індексації, які присутні у традиційних рішеннях для резервного копіювання.

Локальні резервні копії, у свою чергу, передбачають розміщення даних на зовнішніх жорстких дисках або стрічкових системах, які знаходяться в центрі обробки даних або неподалік. Передача даних відбувається через безпечне високошвидкісне з'єднання, пов'язане з глобальною мережею або внутрішньою корпоративною мережею [20].

Варіанти резервного копіювання включають як можливість вручну скопіювати дані на інше місце зберігання, так і використання автоматизованих програм резервного копіювання. Хоча методи резервного копіювання можуть різнитися в залежності від конкретного програмного забезпечення, існують три основні типи, які є загальноприйнятими в більшості цих програм: повне, диференціальне та інкрементне резервне копіювання. Ці типи резервного копіювання визначають механізм копіювання даних з оригінального джерела до місця зберігання і формують підхід до структуризації та зберігання резервних копій.

Повне резервне копіювання, яке включає всі дані в обраних директоріях та файлах, є базовим для всіх інших типів резервного копіювання. Оскільки повне резервне копіювання архівує кожен файл і директорію, воно дозволяє найшвидше та найлегше відновлення даних із резервної копії. На рис. 2.1 можна побачити, як регулярне повне резервне копіювання сприяє збільшенню обсягу резервної копії.

Рис. 2.1. Приклад повного резервного копіювання

З точки зору зручності відновлення даних, постійне проведення повних резервних копій є оптимальним вибором, оскільки це гарантує найвищу повноту та самодостатність збережених даних. Однак, такий підхід має значний недолік – велику витрату часу на створення кожної копії.

Безпека резервних копій також є критично важливим аспектом. Враховуючи, що кожна повна резервна копія включає всі важливі для бізнесу дані, ризик їх незаконного доступу, втрати або крадіжки високий. Тому забезпечення належного рівня захисту резервних копій є таким же важливим, як і захист основних виробничих серверів.

Основні переваги повного резервного копіювання включають:

– швидке відновлення даних порівняно з іншими типами резервного копіювання;

– легкий доступ до останньої резервної копії;

– всі скопійовані дані зосереджені в одному файлі, що спрощує управління ними.

Але є й недоліки, серед яких виділяються:

– повне резервне копіювання є найбільш часомістким у порівнянні з іншими методами;

– вимоги до обсягу місця для зберігання є вищими, ніж у випадку з інкрементним або диференціальним резервним копіюванням.

Диференціальне резервне копіювання – це метод резервного копіювання, при якому копіюються лише ті дані, які змінилися чи були додані після останнього повного резервного копіювання (рис. 2.2). Цей метод не створює копій даних, які залишилися незмінними з моменту останнього повного копіювання, що дозволяє заощадити час та місце для зберігання. Однак для повного відновлення даних знадобиться як остання повна резервна копія, так і останній диференціальний резервний набір.

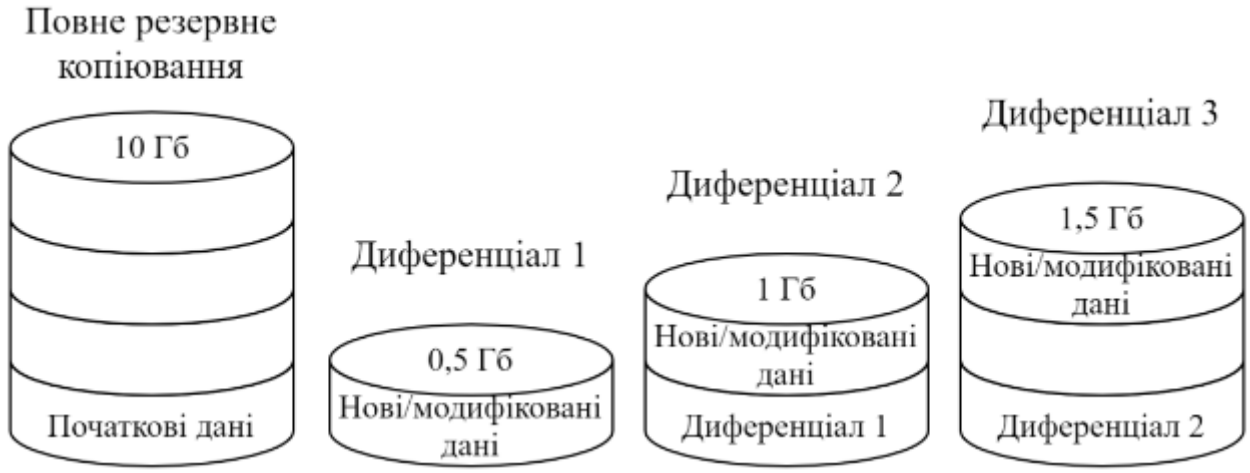

Рис. 2.2. Приклад диференціального резервного копіювання

Диференціальне резервне копіювання, виконане чотири рази, включає наступні етапи. Спершу здійснюється створення повної резервної копії всіх даних. Потім, під час першого диференціального копіювання, робиться копія лише тих даних, які зазнали змін після створення повної копії. Цю першу диференціальну копію називаємо «Диференціал 1», і вона займає 0,5 *Гб*.

Далі, під час наступного диференціального копіювання, створюється «Диференціал 2», що містить усі зміни, що сталися з часу останнього повного копіювання. Ця копія займає 1 *Гб* і включає в себе зміни, які були зафіксовані у «Диференціалі 1».

При четвертому кроці виконується чергове диференціальне копіювання, що охоплює всі зміни з часу останнього повного копіювання, включаючи дані з «Диференціалу 2». Ця копія називається «Диференціал 3».

Диференціальне резервне копіювання має свої переваги та недоліки, які варто враховувати при виборі стратегії резервного копіювання. Переваги даного методу складаються із:

– ефективність використання простору. На відміну від повного резервного копіювання, диференціальне копіювання зберігає лише ті дані, які змінилися з часу останнього повного копіювання. Це дозволяє зменшити обсяг даних, які потрібно зберігати, тим самим економлячи місце на зберігальних пристроях;

– швидше відновлення даних. Під час відновлення даних потрібно лише дві копії - остання повна резервна копія та остання диференціальна копія. Це спрощує та прискорює процес відновлення порівняно з інкрементним резервним копіюванням, де потрібно використовувати кілька наборів даних;

– підвищення надійності. Оскільки кожна диференціальна копія містить усі зміни з моменту останнього повного копіювання, немає ризику втратити критичні дані через відсутність будь-якої окремої інкрементної копії, що робить процес більш надійним.

До недоліків цього методу відносяться:

– збільшення обсягу з кожним копіюванням. Оскільки кожна диференціальна копія включає всі зміни з часу останнього повного копіювання, обсяг даних, який вона займає, зростає з кожним копіюванням, що може призвести до великого використання сховища даних з часом;

– повільніше резервне копіювання з часом. По мірі накопичення змінений та нових даних, диференціальне копіювання може ставати все повільнішим, адже кожен раз необхідно копіювати більший обсяг даних;

– ризик втрати даних при втраті останньої копії. У випадку втрати останньої диференціальної копії, втрачається можливість повного відновлення даних, оскільки вона містить всі зміни, що відбулися після останнього повного резервного копіювання.

Інкрементальне резервне копіювання — це метод резервного копіювання, при якому копіюються лише ті дані, які зазнали змін після останнього резервного копіювання, незалежно від того, чи було це повне чи попереднє інкрементальне

копіювання (рис. 2.3). Цей метод мінімізує обсяг даних, що потребують копіювання на кожному етапі, забезпечуючи ефективність та зниження часу, необхідного для виконання копіювання.

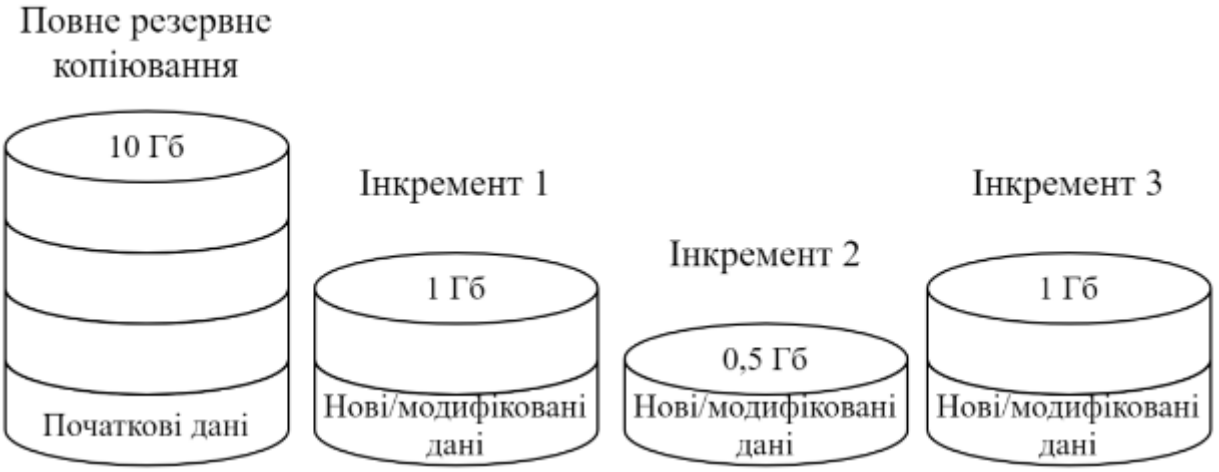

Рис. 2.3. Приклад інкрементального резервного копіювання

Перевагами інкрементального резервного копіювання є:

– ефективність використання простору. Інкрементальне копіювання зберігає лише ті дані, які змінились з часу останнього резервного копіювання, що значно зменшує обсяг необхідного місця для зберігання порівняно з повним копіюванням;

– швидкість копіювання. Цей метод є швидшим, оскільки копіюються лише змінені або нові файли, а не весь обсяг даних;

– зменшення навантаження на мережу і системи. Інкрементальне копіювання вимагає меншого обсягу даних для передачі, що знижує навантаження на мережу і системи зберігання.

Недоліками є:

– складність відновлення даних. Для повного відновлення даних необхідно мати доступ до всіх інкрементних копій та останньої повної копії, що може бути часомістким і складним процесом;

– ризик втрати даних. Якщо одна з інкрементних копій пошкоджена або втрачена, відновлення всіх даних може стати неможливим;

– непостійність використання місця для зберігання. Хоча інкрементальне копіювання спочатку вимагає менше місця для зберігання, з часом обсяг збережених даних може значно зрости, особливо якщо джерельні дані часто змінюються [21].

2.2. Особливості резервного копіювання великих обсягів даних

Резервне копіювання великих обсягів даних у хмарних сервісах вимагає врахування низки ключових аспектів, щоб гарантувати ефективність, безпеку та доступність даних. Вибір правильної стратегії та інструментів є критично важливим для успішного управління даними в сучасному цифровому просторі. З огляду на ці виклики, важливо зосередитися на таких ключових аспектах резервного копіювання в хмарі:

– масштабованість сховища. Хмарні сервіси надають майже необмежену можливість збільшення обсягу сховища, дозволяючи компаніям зберігати великі масиви даних без необхідності інвестувати в дороге фізичне обладнання;

– гнучкість ресурсів та оплати. Користувачі платять лише за використане сховище або обчислювальні ресурси, що дозволяє оптимізувати витрати відповідно до поточних потреб;

– автоматизація процесів. Хмарні сервіси забезпечують можливості автоматизації рутинних процесів резервного копіювання, зменшуючи ризик людської помилки та витрати часу;

– швидкість передачі даних. Залежно від провайдера та конфігурації, хмарні сервіси можуть пропонувати високу швидкість передачі даних, що є важливим для ефективного резервного копіювання великих обсягів;

– безпека та шифрування. Хмарні сервіси часто включають розширені можливості захисту даних, включаючи шифрування даних під час передачі та зберігання;

– дедуплікація даних. Для ефективного використання сховища, багато хмарних сервісів використовують дедуплікацію, що дозволяє уникнути зберігання дублікатів даних.

Діаграма у вигляді ментальної карти, яка ілюструє особливості резервного копіювання великих обсягів даних у хмарних сервісах представлена на рис. 2.4.

Рис. 2.4. Особливість резервного копіювання у хмарних сервісах

Резервне копіювання великих обсягів даних у хмарні сервіси - це комплексний процес, який включає кілька ключових етапів. Починається все з оцінки та планування, де аналізуються дані для визначення їх типу, розміру, частоти змін та важливості. Потім відбувається вибір хмарного провайдера, що відповідає вимогам до обсягу, безпеки, вартості та процесам копіювання. Після цього наступає фаза реалізації, яка включає налаштування інфраструктури, застосування процедур дедуплікації та оптимізації даних, а також встановлення автоматизованих процедур для регулярного резервного копіювання.

Безпека є критично важливою, тому використовується сильне шифрування даних та ретельне керування доступом для запобігання несанкціонованому доступу.

Важливою частиною процесу є відновлення та тестування, яке включає розробку стратегій відновлення даних для аварійних ситуацій та проведення регулярних тестів для перевірки надійності та ефективності резервних копій.

Останній етап - моніторинг та оптимізація, що передбачає неперервний моніторинг процесів резервного копіювання та постійне оновлення та оптимізація

процесів і технологій для забезпечення максимальної ефективності та відповідності поточним вимогам [22].

2.3. Аналіз ефективності існуючих рішень для резервного копіювання в хмарних середовищах

Аналіз ефективності існуючих рішень для резервного копіювання в хмарних середовищах є ключовим для розуміння їхньої придатності в різних бізнес-сценаріях та індивідуальних потребах. У цьому аналізі увага зосереджена на трьох популярних сервісах: *Google Drive, Microsoft Azure ma Amazon Web Services*. Кожен з цих сервісів має свої унікальні особливості, переваги та недоліки, які варто розглянути для обрання найоптимальнішого рішення.

Google Drive – це хмарний сервіс для зберігання та синхронізації файлів, розроблений компанією *Google* (рис. 2.5). Запущений у 2012 році, *Google Drive* дозволяє користувачам зберігати файли на їхніх серверах, синхронізувати файли між різними пристроями та ділитися ними. Окрім зберігання даних, *Drive* інтегрований з *Google Docs*, *Sheets* та *Slides*, набором офісних інструментів для спільної роботи над документами, електронними таблицями та презентаціями.

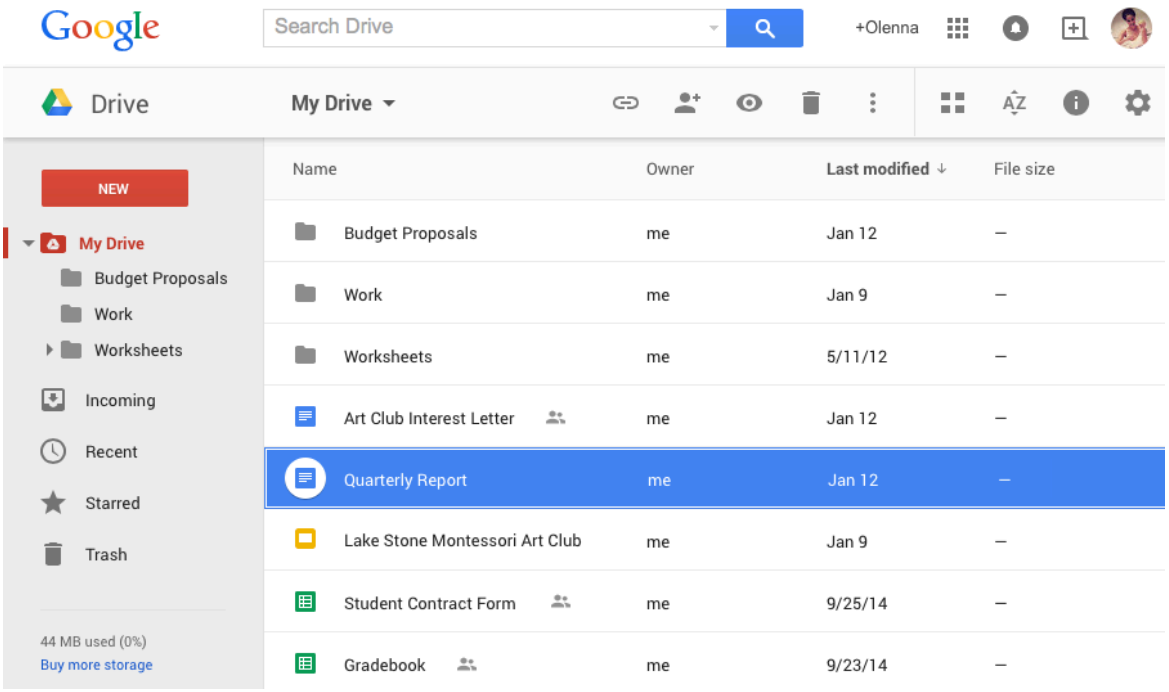

Рис. 2.5. Приклад інтерфейсу «*Google Drive*»

Архітектура *Google Drive* базується на хмарних технологіях і використовує розподілені сервери *Google* для зберігання даних. Основні аспекти архітектури:

– хмарне зберігання. Дані зберігаються на серверах *Google*, що забезпечує високу доступність та надійність;

– сервери *Google*. Дані розподіляються між численними центрами обробки даних, що знижує ризик втрати даних та підвищує швидкість доступу;

– клієнтські додатки. *Google Drive* пропонує додатки для різних платформ, включаючи *Windows*, *macOS*, *Android* та *iOS*, для синхронізації та доступу до файлів;

– *API*. *Google Drive* має *API*, що дозволяє інтеграцію з різними додатками та сервісами, розширюючи функціональність.

Загалом, *Google Drive* є універсальним рішенням для зберігання даних, яке забезпечує гнучкість, безпеку та простоту використання для індивідуальних користувачів та бізнесу. Основними перевагами *Google Drive* є:

– інтеграція з *Google Workspace*. Тісно інтегрований з іншими сервісами *Google*, такими як *Docs*, *Sheets* та *Slides*, що дозволяє легко створювати, редагувати та спільно працювати над документами в реальному часі;

– легкість доступу та синхронізації. Завдяки додаткам для різних платформ, таких як *Windows*, *macOS*, *Android* та *iOS*, користувачі можуть легко доступатися до своїх файлів з будь-якого пристрою та синхронізувати їх між різними пристроями;

– безпека даних. *Google Drive* забезпечує високий рівень безпеки для зберіганих файлів, використовуючи передові технології шифрування та надійні протоколи безпеки;

– гнучкі опції зберігання. Початковий безкоштовний план надає достатній обсяг сховища для більшості індивідуальних користувачів, а платні плани дозволяють легко розширити обсяг зберігання згідно з потребами.

Недоліки *Google Drive*:

– обмеження безкоштовного сховища. Безкоштовний план має обмеження на обсяг сховища, що може бути недостатнім для користувачів з великою кількістю даних;

– приватність даних. Хоча *Google* забезпечує високу безпеку, існують побоювання щодо приватності, оскільки компанія аналізує дані для надання персоналізованих реклам та інших цілей;

– комплексність деяких функцій. Для нових користувачів деякі функції *Google Drive* можуть здатися складними або неінтуїтивними, особливо коли йдеться про спільне використання файлів та управління правами доступу.

Отже, *Google Drive* є ефективним та широко доступним хмарним рішенням для зберігання, синхронізації та спільної роботи над файлами. Його тісна інтеграція з *Google Workspace*, включаючи *Docs*, *Sheets* та *Slides*, робить його ідеальним вибором для співпраці та продуктивності. Завдяки зручним додаткам для різноманітних платформ, *Google Drive* забезпечує легкий доступ до файлів з будь-якого місця та пристрою. Однак, існують певні недоліки, як-от обмежений безкоштовний обсяг сховища та побоювання щодо приватності даних, а також залежність від стабільного Інтернет-з'єднання. Деякі користувачі також можуть знайти інтерфейс та функції складними для освоєння [23].

Amazon Web Services (*AWS*) - це широко використовуваний хмарний платформний сервіс, запущений компанією *Amazon* у 2006 році. *AWS* надає широкий спектр хмарних сервісів, включаючи обчислювальні потужності, зберігання даних, бази даних та інструменти для розробки.

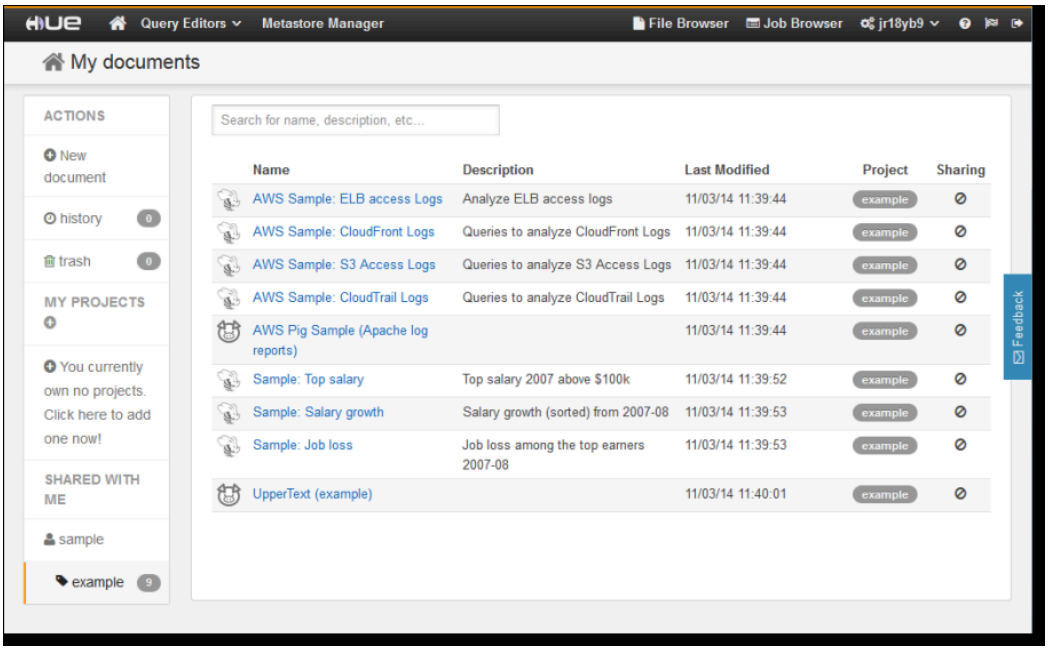

Рис. 2.6. Приклад інтерфейсу «*Amazon Web Services*»

Архітектура *AWS* базується на хмарній інфраструктурі, розподіленій по всьому світу у вигляді «регіонів» та «зон доступності».

Переваги *Amazon Web Services*:

– масштабованість та еластичність. *AWS* дозволяє користувачам легко масштабувати ресурси згідно з їхніми потребами, забезпечуючи високу гнучкість в управлінні навантаженням і потребами в обчислювальних ресурсах;

– надійність та висока доступність. Завдяки глобально розподіленій інфраструктурі, що включає численні регіони та зони доступності, *AWS* забезпечує високий рівень надійності та доступності своїх сервісів;

– широкий спектр сервісів. *AWS* пропонує велику кількість сервісів, що покривають майже всі потреби в хмарних обчисленнях, включаючи обчислювальні потужності, зберігання даних, бази *даних*, машинне навчання та багато іншого;

– безпека та комплаєнс. *AWS* надає розширені можливості щодо безпеки, включаючи шифрування, управління ідентифікацією та доступом, дотримуючись при цьому строгих стандартів комплаєнсу.

Недоліки *Amazon Web Services*:

– складність управління. Широкий спектр сервісів та опцій може ускладнити управління та конфігурацію, особливо для нових користувачів або тих, хто не має глибоких технічних знань;

– витрати. Хоча *AWS* пропонує гнучкі варіанти ціноутворення, іноді може бути важко передбачити загальні витрати, особливо коли використовуються численні сервіси та ресурси;

– залежність від одного провайдера. Застосування *AWS* може призвести до залежності від одного хмарного провайдера, що може створити ризики для бізнесу в плані гнучкості та витрат;

– запитання щодо приватності та юрисдикції даних. Деякі користувачі можуть мати занепокоєння щодо приватності та юрисдикції даних, оскільки всі дані зберігаються на серверах *AWS*, які можуть знаходитися в різних країнах із різними законодавчими нормами.

Amazon Web Services пропонує потужну, масштабовану та надійну платформу, яка підходить для широкого спектру бізнес-потреб. Завдяки глобальній мережі регіонів та зон доступності, *AWS* забезпечує високу доступність та надійність своїх сервісів, а також пропонує широкий вибір інструментів та сервісів, від обчислювальних потужностей до баз даних та інструментів машинного навчання.

Однак, з огляду на складність платформи та її витрати, *AWS* може бути недостатньо простим для нових користувачів або для тих, хто не має глибоких технічних знань. Крім того, залежність від одного хмарного провайдера може створювати певні ризики для бізнесу. Незважаючи на це, *AWS* продовжує бути одним з найбільш популярних та широко використовуваних хмарних рішень у світі, пропонуючи високий рівень безпеки, надійності та інноваційних технологій для підприємств усіх розмірів [24].

Microsoft Azure є хмарною платформою від *Microsoft*, запущеною у 2010 році, яка надає широкий спектр хмарних сервісів, включаючи обчислення, зберігання даних, бази даних, мережеві рішення та інші (рис. 2.7). Призначена для підприємств та розробників, дозволяє створювати, тестувати, запускати та управляти додатками та сервісами через глобально розподілену мережу центрів обробки даних *Microsoft*.

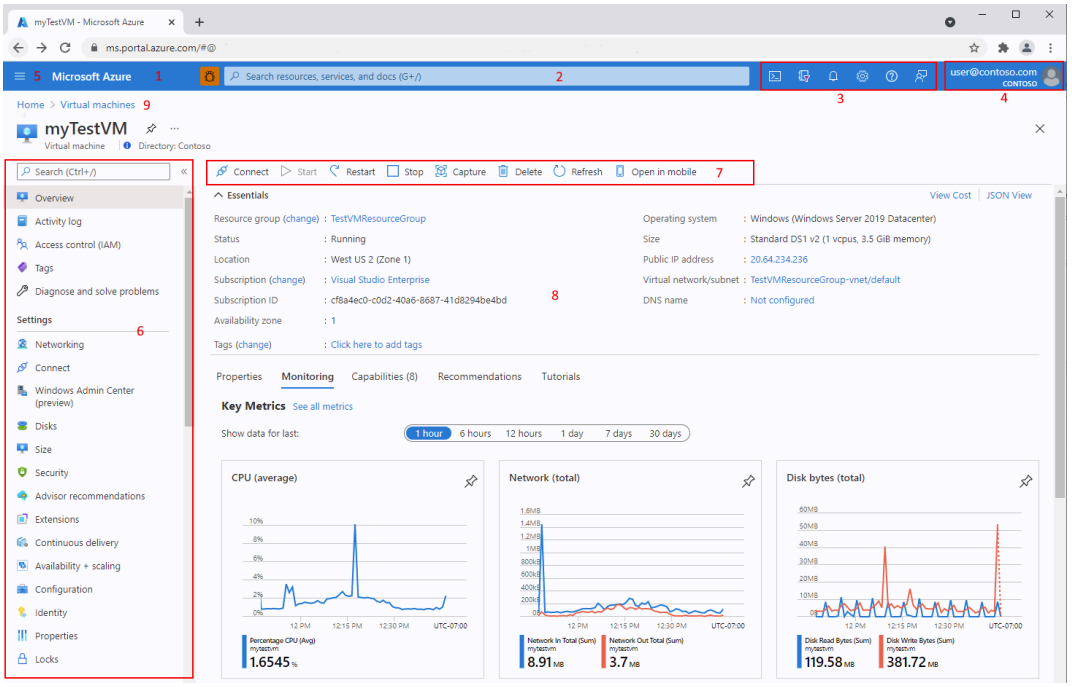

Рис. 2.7. Приклад інтерфейсу «*Microsoft Azure*»

Архітектура *Azure* базується на хмарній інфраструктурі, що забезпечує високу масштабованість та гнучкість. Вона включає численні центри обробки даних по всьому світу, розділені на регіони та зони доступності для забезпечення високої надійності та безперервності сервісу. *Azure* надає різноманітність сервісів та рішень, які підтримуються широким спектром технологій, включаючи машинне навчання, штучний інтелект, Інтернет речей та багато інших, дозволяючи підприємствам інноваційно розвиватися в хмарному середовищі.

Переваги *Microsoft Azure*:

– інтеграція з іншими продуктами *Microsoft*. Легко інтегрується з широким спектром інших продуктів *Microsoft*, таких як *Windows Server*, *Active Directory*, та *SQL Server*, що забезпечує плавний перехід для багатьох компаній;

– підтримка гібридних хмарних рішень. Надає сильну підтримку гібридних хмар, дозволяючи компаніям поєднувати використання хмарних та локальних ресурсів, забезпечуючи гнучкість та оптимізацію інфраструктури;

– висока масштабованість та гнучкість. Дозволяє користувачам швидко масштабувати ресурси вгору або вниз залежно від потреб, пропонуючи високу гнучкість у управлінні ресурсами;

– широкий спектр сервісів та рішень. Пропонує велике різноманіття сервісів та рішень, включаючи машинне навчання, аналітику даних, Інтернет речей, та багато інших, забезпечуючи компаніям інструменти для інноваційного розвитку.

Недоліками *Microsoft Azure* є:

– складність управління та налаштування. Через велику кількість сервісів та можливостей, *Azure* може бути складним у налаштуванні та управлінні, особливо для нових користувачів або тих, хто не має глибоких технічних знань;

– вартість. Хоча *Azure* пропонує конкурентоспроможні ціни, витрати можуть швидко зрости, особливо при використанні розширених сервісів або великому обсязі ресурсів.

– запитання щодо приватності та безпеки даних. Деякі компанії можуть мати занепокоєння щодо зберігання даних в хмарі, особливо у контексті приватності та безпеки даних, а також щодо дотримання регуляторних вимог.

Отже, *Microsoft Azure* представляє собою потужну та гнучку хмарну платформу, яка ідеально підходить для широкого спектру бізнес-завдань, від розгортання веб-додатків до обробки великих даних та машинного навчання. Її здатність до інтеграції з іншими продуктами *Microsoft* та підтримка гібридних хмарних рішень робить її привабливою для багатьох компаній, які вже використовують інфраструктуру *Microsoft*. Масштабованість та різноманітність сервісів *Azure* надають користувачам гнучкість та можливості для інноваційного розвитку.

Однак, складність управління та висока вартість можуть бути перешкодами, особливо для малих та середніх підприємств або для тих, хто лише починає працювати з хмарними технологіями. Також питання приватності та безпеки даних є важливими аспектами для розгляду при виборі хмарного провайдера. В цілому, *Azure* є важливим гравцем на ринку хмарних сервісів, пропонуючи міцну платформу для бізнесів, які прагнуть використовувати переваги хмарних технологій [25].

2.4. Інтеграція систем резервного копіювання з хмарними сервісами

Інтеграція застосунків із хмарними сервісами, такими як *Google Drive*, включає використання різних технік і технологій для ефективного з'єднання із програмами з хмарними можливостями зберігання та обробки даних. Основним інструментом для інтеграції є *Google Drive API*, який дозволяє застосункам взаємодіяти з файлами та папками на *Google Drive*.

Перший крок у процесі інтеграції полягає в створенні проекту в *Google Cloud Console* та отриманні відповідних ключів доступу, таких як *OAuth 2.0 client IDs*. Це забезпечує безпечну авторизацію та аутентифікацію застосунку при взаємодії з *Google Drive*. Після налаштування аутентифікації, можна використовувати *Google Drive API* для виконання різних операцій, таких як завантаження файлів на *Drive*, отримання списку файлів, створення нових папок, пошук файлів, а також для редагування та видалення існуючих файлів. *API* підтримує різні мови

програмування, включаючи *Java*, *Python*, *JavaScript* та інші, що надає гнучкість у виборі технологій для розробки застосунків.

Окрім того, інтеграція з *Google Drive* може включати використання сервісів *Google Cloud* для підвищення масштабованості та ефективності. Наприклад, можна використовувати *Google Cloud Functions* для автоматизації завдань, пов'язаних з обробкою файлів на *Drive*, або *Google Cloud Pub/Sub* для управління подіями, що виникають у *Drive*. Також важливо забезпечити, що інтеграція відповідає стандартам безпеки та конфіденційності даних. Це включає належне керування правами доступу до файлів та папок, а також забезпечення захисту даних під час їх передачі та зберігання.

На рис. 2.8 представлено діаграму послідовності взаємодії компонентів системи і її інтеграція із хмарним сервісом *Google Drive*.

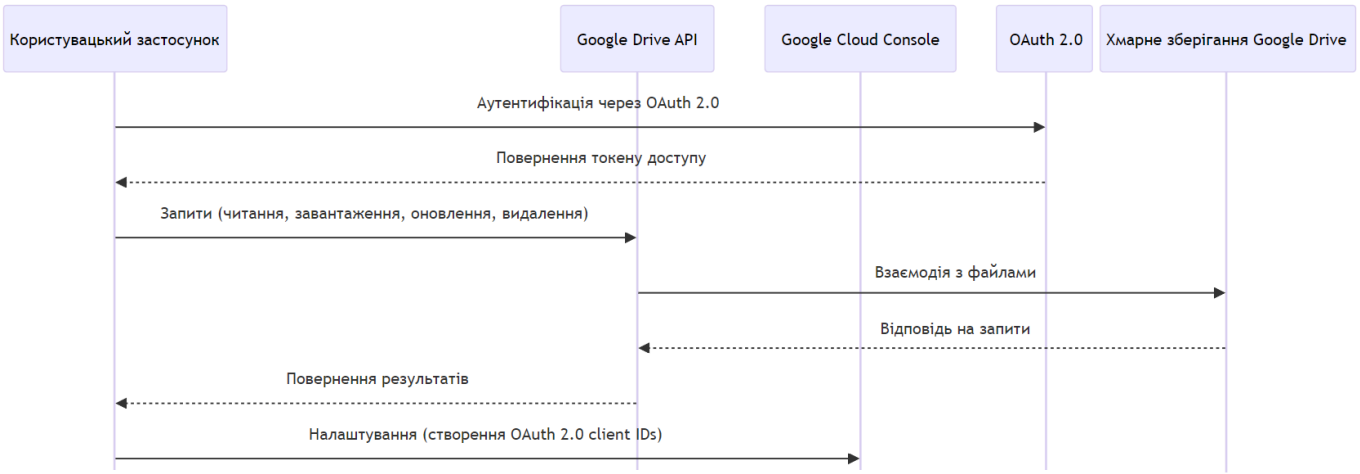

Рис. 2.8. Інтеграція застосунків із «*Google Drive*»

У процесі інтеграції, застосунок спочатку використовує *OAuth 2.0* для аутентифікації користувача та отримання дозволу на доступ до їхніх файлів. Після успішної аутентифікації, застосунок може відправляти запити до *Google Drive API*, використовуючи токен доступу. *API* тоді взаємодіє з хмарним зберіганням *Google Drive* для виконання цих запитів, і повертає відповіді назад до застосунку.

Інтеграція застосунків із хмарними сервісами *AWS* вимагає розуміння та використання специфічних хмарних ресурсів та сервісів, які *AWS* пропонує. Основними компонентами цього процесу є *AWS SDK (Software Development Kit)*, *AWS API Gateway, IAM (Identity and Access Management)* для безпеки та авторизації, а

також різноманітні хмарні сервіси, як-от *EC2* для віртуальних серверів, *S3* для зберігання, *Lambda* для безсерверних обчислень тощо.

Розробники спочатку використовують *AWS SDK*, який доступний для багатьох мов програмування (наприклад, *Java*, *Python, .NET*), для інтеграції їхніх застосунків із *AWS*. *SDK* надає інструменти та бібліотеки для спрощення взаємодії з *AWS* сервісами. Завдяки цьому, застосунки можуть легко викликати *API* хмарних сервісів, управляти ресурсами та збирати дані.

AWS API Gateway використовується для створення, публікації, підтримки, моніторингу та захисту *API* на будь-якій шкалі. Це дає можливість розробникам створювати вхідні точки для доступу до функціональності їхніх застосунків. Щодо безпеки, *AWS IAM* використовується для управління доступом до *AWS* ресурсів. Розробники можуть визначити, хто може взаємодіяти з ресурсами *AWS*, які дії вони можуть виконувати, і яким чином. Це забезпечує високий рівень безпеки та контролю доступу.

Щодо хмарних сервісів, *Amazon EC2* надає масштабовані віртуальні сервери, *Amazon S3* пропонує зберігання об'єктів, *Amazon RDS* та *DynamoDB* надають рішення для управління базами даних, а *AWS Lambda* дозволяє запускати код без проведення та управління серверами. Інтеграція також часто включає використання *AWS CloudFormation* для автоматизації розгортання інфраструктури та сервісів. Розробники можуть описати всі необхідні *AWS* ресурси та їхні відносини у *JSON* або *YAML* форматі, що дозволяє легко створювати та управляти повним стеком ресурсів *AWS*.

На рис. 2.9 представлено діаграму послідовності взаємодії компонентів системи і її інтеграція із хмарним сервісом *AWS*.

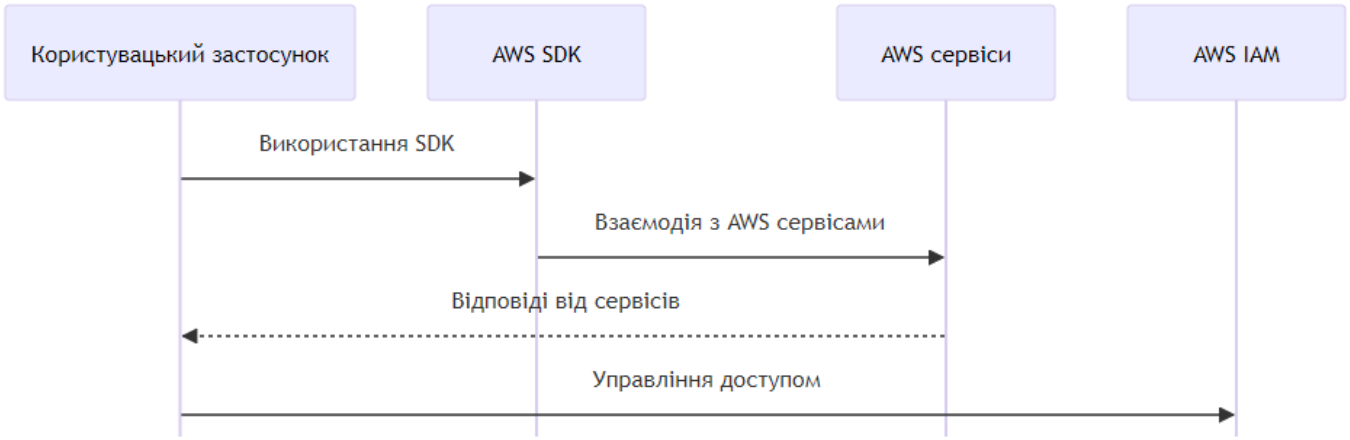

Рис. 2.9. Інтеграція застосунків із «*AWS*»

У цій схемі застосунок взаємодіє з *AWS SDK* та *API Gateway* для виклику різних *AWS* сервісів. *IAM* використовується для управління доступом та авторизацією, в той час як *CloudFormation* дозволяє автоматизувати розгортання та управління інфраструктурою. *ELB* та *Amazon VPC* забезпечують масштабування, балансування навантаження та безпеку мережі. Ці компоненти спільно працюють, забезпечуючи ефективну та безпечну інтеграцію застосунку з хмарними ресурсами *AWS*.

Інтеграція застосунку з сервісами *Microsoft Azure* включає кілька ключових етапів, які забезпечують взаємодію між застосунком та хмарними сервісами *Azure*. Зазвичай така інтеграція відбувається наступним чином:

– створення та налаштування *Azure* ресурсів. Першим кроком є створення та налаштування необхідних ресурсів в *Azure*. Це може включати створення *Azure App Service* для веб-додатків, *Azure Storage* для зберігання даних, *Azure SQL Database* для баз даних та інші сервіси, відповідно до потреб застосунку;

– налаштування *Azure Active Directory* для управління доступом. Щоб забезпечити безпечний доступ до ресурсів *Azure*, необхідно налаштувати *Azure Active Directory (AD)*. Це включає створення облікових записів користувачів, груп, а також налаштування політик доступу та ролей;

– розробка та налаштування застосунку для взаємодії з *Azure*. Застосунок має бути налаштований для взаємодії з *Azure* сервісами. Це зазвичай включає використання *Azure SDK*, який спрощує роботу з *Azure API*. Розробники

використовують *SDK* для написання коду, який взаємодіє з *Azure* сервісами, наприклад, для завантаження файлів у сховище або виконання запитів до бази даних;

– інтеграція через *Azure API Management*. Якщо застосунок використовує або надає *API*, *Azure API Management* може бути використаний для створення та управління *API*. Це дозволяє контролювати, як зовнішні та внутрішні користувачі взаємодіють з *API*, а також забезпечує моніторинг, автентифікацію та шифрування;

– використання *Azure DevOps* для безперервної інтеграції та доставки. *Azure DevOps* може бути використаний для автоматизації процесу розробки, тестування та розгортання застосунку. Це включає налаштування конвеєрів *CI*/*CD*, які автоматично розгортають оновлення та зміни в *Azure*.

Після розгортання застосунку, використовуються інструменти *Azure* для моніторингу його роботи та управління ним. Це може включати *Azure Monitor* для відстеження продуктивності та *Azure Application Insights* для аналізу використання застосунку.

На рис. 2.10 представлена схема послідовності, що ілюструє інтеграцію застосунку з *Microsoft Azure*, включаючи взаємодію між ключовими компонентами.

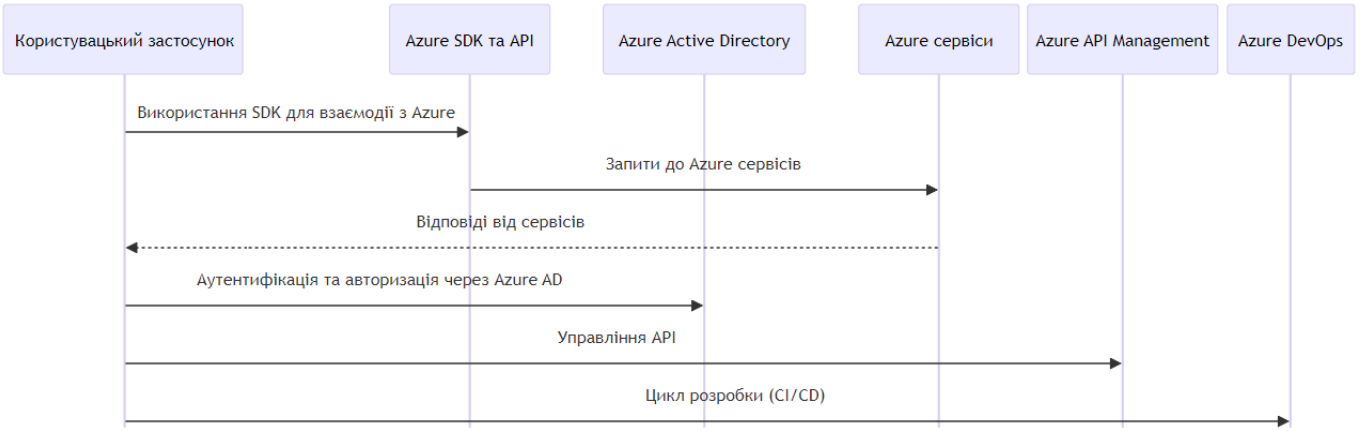

Рис. 2.10. Інтеграція застосунків із «*Microsoft Azure*»

У даній схемі застосунок спілкується з *Azure SDK/API* для взаємодії з різними *Azure* сервісами. *Azure AD* забезпечує аутентифікацію та авторизацію. *API Management* допомагає управляти *API,* а *Azure DevOps* підтримує розробку та розгортання. Моніторинг та аналітика виконуються через *Azure Monitor* та *Application Insights*.

У табл. 2.1 проведено порівняння характеристик інтеграції застосунків з хмарними сервісами *Google Drive*, *Microsoft Azure* та *Amazon Web Services*.

Таблиця 2.1

Аналіз інтеграції із хмарними сервісами

Характеристика	*Google Drive*	*Microsoft Azure*	*Amazon Web Services*
1	2	3	4
Легкість інтеграції	Проста	Складна	Складна
Набір доступних *API*	Обмежений	Широкий	Широкий

Закінчення таблиці 2.1

1	2	3	4
Можливості зберігання даних	Високі можливості для файлів та документів	Широкі можливості зберігання та обробки даних	Широкі можливості зберігання та обробки даних
Безпека	Висока	Висока	Висока
Підтримка різних мов програмування	Обмежена (залежить від *Google API*)	Широка (підтримка багатьох мов)	Широка (підтримка багатьох мов)
Автоматизація та управління	Обмежена	Розширена (*Azure DevOps*)	Розширена (*AWS Lambda, AWS CloudFormation*)
Спеціалізовані хмарні сервіси	Обмежені	Розширені (великий вибір сервісів)	Розширені (великий вибір сервісів)
Екосистема та інтеграція з іншими сервісами	Висока (інтеграція з *Google Workspace*)	Середня (інтеграція з *Microsoft* продуктами)	Середня

Вибір *Google Drive* для розробки системи управління та резервного копіювання даних можна обґрунтувати декількома ключовими факторами:

– простота інтеграції. *Google Drive* дозволяє легко інтегруватися з різними застосунками та сервісами. Це означає, що розробники можуть швидко налаштувати систему управління та резервного копіювання без необхідності розбиратися в складних хмарних конфігураціях;

– надійність та доступність. Завдяки інфраструктурі *Google*, *Drive* гарантує високу доступність та надійність зберігання даних. Це важливо для систем резервного копіювання, де втрата даних або їх недоступність може мати серйозні наслідки;

– інтеграція з *Google Workspace*. Якщо система буде використовуватися в середовищі, де вже активно використовуються інші продукти *Google*, такі як *Google Docs*, *Sheets*, то інтеграція з *Google Drive* стане природним рішенням, яке забезпечує плавну сумісність;

– легкість спільного використання та співпраці. *Google Drive* дозволяє легко ділитися файлами та папками, забезпечуючи можливості співпраці, що є корисним для команд, які працюють з даними;

– безпека та контроль доступу. *Google Drive* надає різні опції для контролю доступу до файлів та папок, забезпечуючи безпеку зберіганих даних. Це включає шифрування даних під час передачі та зберігання.

Отже, *Google Drive* пропонує гарне поєднання простоти, функціональності та надійності для систем управління та резервного копіювання даних, особливо для організацій, які шукають просте та ефективне рішення.

2.5. Висновки до розділу

У рамках даного розділу було проведено детальний аналіз методів резервного копіювання великих обсягів даних. Основну увагу приділено порівнянню традиційних та сучасних стратегій, включаючи їхні переваги, недоліки, та особливості застосування в хмарних сервісах. Було детально розглянуто повне, диференційне та інкрементальне резервне копіювання, їхні ключові характеристики та приклади застосування. Також, висвітлено особливості резервного копіювання в

хмарних сервісах, зокрема масштабованість сховища, гнучкість ресурсів, автоматизацію процесів, швидкість передачі даних, безпеку, та дедуплікацію даних. Здійснено аналіз ефективності існуючих рішень для резервного копіювання в хмарних середовищах, зокрема у *Google Drive, Microsoft Azure*, та *Amazon Web Services*. Окремо було розглянуто інтеграцію систем резервного копіювання з хмарними сервісами, включаючи технічні особливості та методи інтеграції.

У результаті проведеного аналізу ефективності існуючих рішень для резервного копіювання в хмарних середовищах було визначено, що *Google Drive* є найбільш оптимальним варіантом. Вибір *Google Drive* обумовлений його зручністю у використанні, високим рівнем інтеграції з іншими сервісами *Google*, а також ефективними механізмами синхронізації та спільного доступу до даних, що сприяє більшій гнучкості та оперативності процесу резервного копіювання.

РОЗДІЛ 3
ПРОЕКТУВАННЯ ТА АНАЛІЗ СИСТЕМИ УПРАВЛІННЯ ТА РЕЗЕРВНОГО КОПІЮВАННЯ

3.1. Формулювання вимог до системи

Впровадження системи електронного документообігу вимагає як визначення специфічних користувацьких потреб, так і розробки функціональності для забезпечення безпеки документів. Ефективність, зручність та надійність системи є ключовими. Важливим етапом є використання *use-case* діаграм для візуалізації взаємодій користувачів із системою, що сприяє розробці функціональної бази. Також, структурування зібраних вимог через специфікацію є основою для розробки та подальшої імплементації системи, гарантуючи її ефективне використання (табл. 3.1).

Таблиця 3.1

Функціональні вимоги до системи

Вимоги	Опис
REQ-1	Забезпечує ідентифікацію та аутентифікацію користувачів, включаючи перевірку прав доступу для збереження конфіденційності та цілісності інформації
REQ-2	Ведення журналу дій користувачів та системних подій, необхідне для аудиту та контролю діяльності в системі
REQ-3	Імплементація системи прав доступу на основі ролей користувачів для регулювання доступу до документів та ресурсів
REQ-4	Інтеграція *з Google Drive* для зберігання документів, підвищення доступності та надійності
REQ-5	Реалізація механізмів пошуку для швидкого доступу до документів
REQ-6	Підтримка версіонування документів для відстеження змін і правок
REQ-7	Впровадження заходів безпеки, включаючи шифрування, *SSL*-сертифікати, та двофакторну автентифікацію

REQ-8	Функція автоматичної генерації звітів для докладного моніторингу діяльності та використання системи

Нефункціональні вимоги оцінюють якість та ефективність системи, зосереджуючись на атрибутах, як-от швидкість реакції, міцність, безпека, адаптація та масштабування. Ці критерії важливі для надійності та безпеки системи, що впливає на її ефективну роботу. В табл. 3.2 детально представлені ці вимоги, даючи цілісне уявлення про їх важливість для продуктивності системи та визначення умов для її успішного впровадження та експлуатації.

Таблиця 3.2

Нефункціональні вимоги до системи

Вимоги	Опис
REQ-9	Система повинна забезпечувати високу швидкодію для швидкої реакції на запити, навіть при значній кількості користувачів і запитів
REQ-10	Стабільність системи з мінімальною кількістю збоїв для забезпечення надійності середовища користувачів
REQ-11	Цілодобовий доступ до репозиторію з мінімальними перервами, гарантуючи неперервність доступу до документів
REQ-12	Ефективні механізми захисту конфіденційної інформації для протидії несанкціонованому доступу та збереження приватності
REQ-13	Інтуїтивно зрозумілий та візуально привабливий інтерфейс користувача для підвищення користувацького досвіду
REQ-14	Оптимізація використання апаратних ресурсів для забезпечення ефективності системи

У розроблюваній системі значну роль відіграють актори, які включають як індивідуальних користувачів, так і групи, взаємодіючи з платформою. Основними серед них є адміністратори та кінцеві користувачі системи. Для кожного актора чітко визначені їхні унікальні ролі, обов'язки, та цілі, важливі для ефективної взаємодії з системою. Ці відомості докладно систематизовані у табл. 3.3, надаючи глибоке розуміння механізму взаємодії кожного актора з системою та їх вплив на її роботу, забезпечуючи огляд всіх ключових аспектів функціонування системи.

Таблиця 3.3

Актори та їхні цілі у системі

Актори	Цілі
Адміністратор	відповідає за управління репозиторієм, контроль прав доступу, захист даних та моніторинг активності користувачів
Користувач	зосереджені на продуктивному використанні репозиторію для управління, збереження, пошуку, завантаження та редагування документів
База даних	слугує для зберігання всієї інформації про документи та їх метадані
Google Drive	забезпечує хмарне зберігання, підвищуючи доступність та надійність

Чітке визначення *use-case* сценаріїв є ключовим для взаємодії учасників із системою управління локальними документами. Кожен сценарій представляє послідовність дій, які актори виконують для розв'язання конкретної задачі. В табл. 3.4 наведено основні сценарії користування системою, детально описуючи процеси завантаження, пошуку, редагування, видалення документів в хмарі.

Таблиця 3.4

Опис варіантів використання системи

№	Ім’я	Опис
UC1	Реєстрація користувача	можливість для новачків створити акаунт, вносячи особисті дані для входу в систему
UC2	Автентифікація в системі	вимога ввести логін та пароль для ідентифікації перед доступом до системи
UC3	Перегляд каталогу користувачів	дозвіл адміністраторам із правами доступу оглядати зареєстрованих користувачів
UC4	Створення користувача	дозволяє адміністратору реєструвати нових користувачів, встановлюючи їх ролі
UC5	Редагування профілю користувача	можливість для адміністратора редагувати інформацію профілів користувачів

UC6	Конфігурація доступу до репозиторію	встановлення адміністратором рівнів доступу до репозиторію

Закінчення таблиці 3.4

UC7	Синхронізація бази даних з репозиторієм	актуалізація інформації у репозиторії, яка може бути автоматичною чи ручною
UC8	Персоналізація	дозволяє користувачу оновити дані свого профілю
UC9	Переглядання документів репозиторію	можливість перегляду всіх документів, що зберігаються в репозиторії
UC10	Завантажування нових файлів	дозволяє користувачам додавати нові файли до репозиторію
UC11	Контроль версій файлів	забезпечує можливість оновлення версій файлів у репозиторії
UC12	Пошук документів	відкриває можливості для пошуку документів за різними параметрами
UC13	Перегляд подій	дозволяє переглядати журнали подій або логи, де записуються важливі події в системі для користувачів та адміністраторів

3.2. Побудова *use-case* діаграми прецедентів

Виходячи з визначених вимог, були створені *use-case* діаграми для системи «Репозиторій електронних документів», що детально відображають взаємодії та завдання учасників, особливо для ролей «Адміністратор» та «Користувач». Діаграма для «Користувача» (рис. 3.1) демонструє різні способи їх взаємодії з системою, як-от завантаження та редагування документів. Діаграма для «Адміністратора» (рис. 3.2) зосереджується на їхніх ключових обов'язках і можливостях у системі.

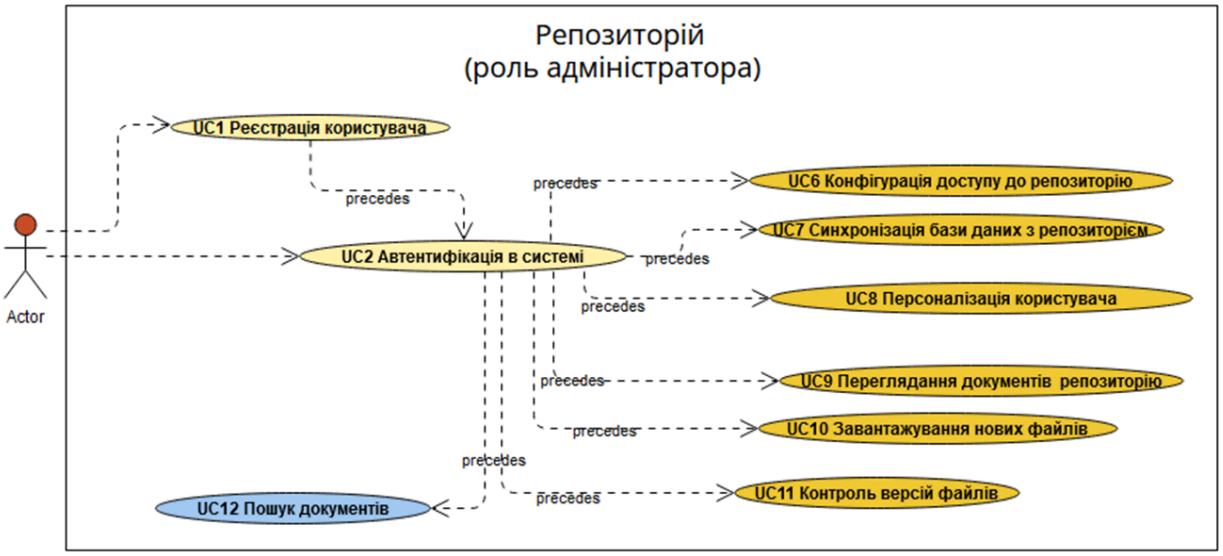

Рис. 3.1. Діаграма прецедентів для ролі «Користувач»

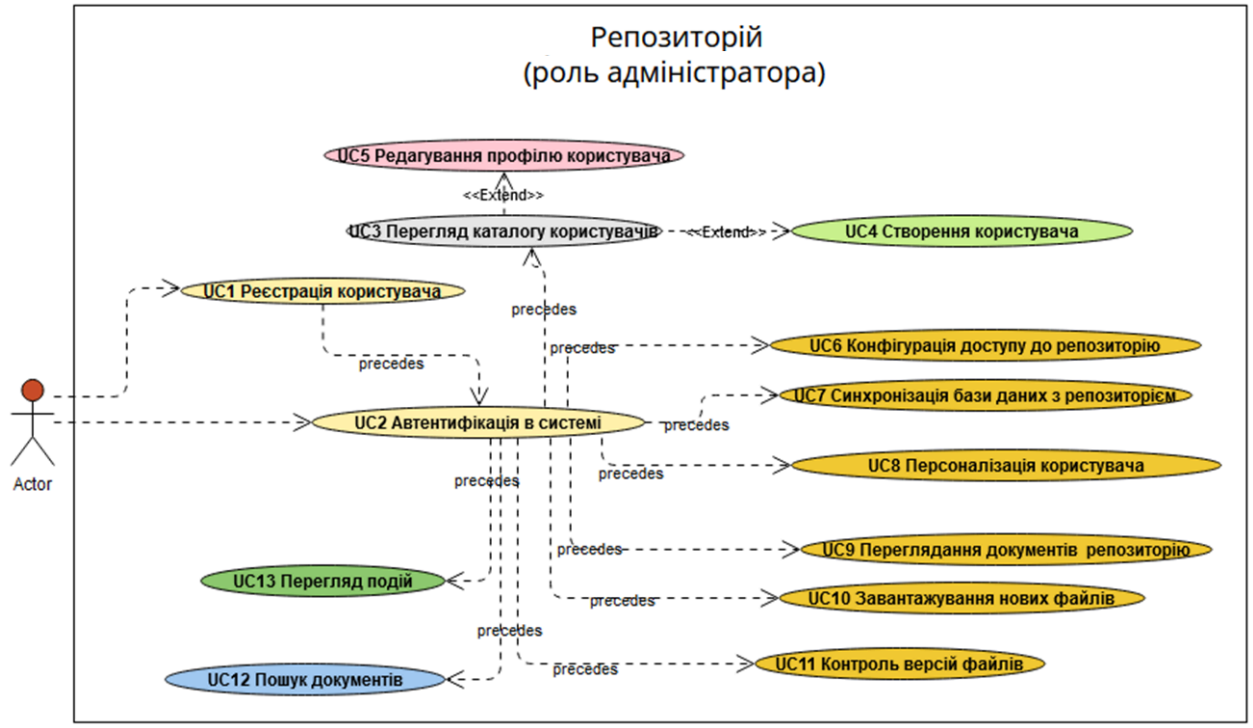

Рис. 3.2. Діаграма прецедентів для ролі «Адміністратор»

Отже, розроблені діаграми прецедентів не лише відображають взаємодії в системі «Репозиторій», а й важливі для зрозуміння її функціональних та нефункціональних елементів на початкових етапах розробки. Ці діаграми сприяють організації проекту, забезпечуючи його гнучкість та адаптацію протягом розробки та впровадження.

3.3. Побудова діаграм бізнес-процесів

Під час проектування системи «Репозиторій» були створені детальні діаграми діяльності та послідовності, які відображають важливі взаємодії всередині системи. Ці діаграми, важливі для системного аналізу, ілюструють залежності між елементами та користувачами системи. Вони включають авторизацію користувачів, взаємодії з документами та інші ключові процеси, що допомагають користувачам та розробникам зрозуміти, як система обробляє запити та впливає на документи.

На одній із діаграм, яка зображена на рис. 3.3, детально представлено процес авторизації користувача в системі. Ця діаграма покроково демонструє дії системи починаючи з моменту, коли користувач натискає кнопку авторизації, і закінчуючи валідацією введених даних, перевіркою користувача в базі даних, та відображенням головного інтерфейсу або повідомлення про помилку.

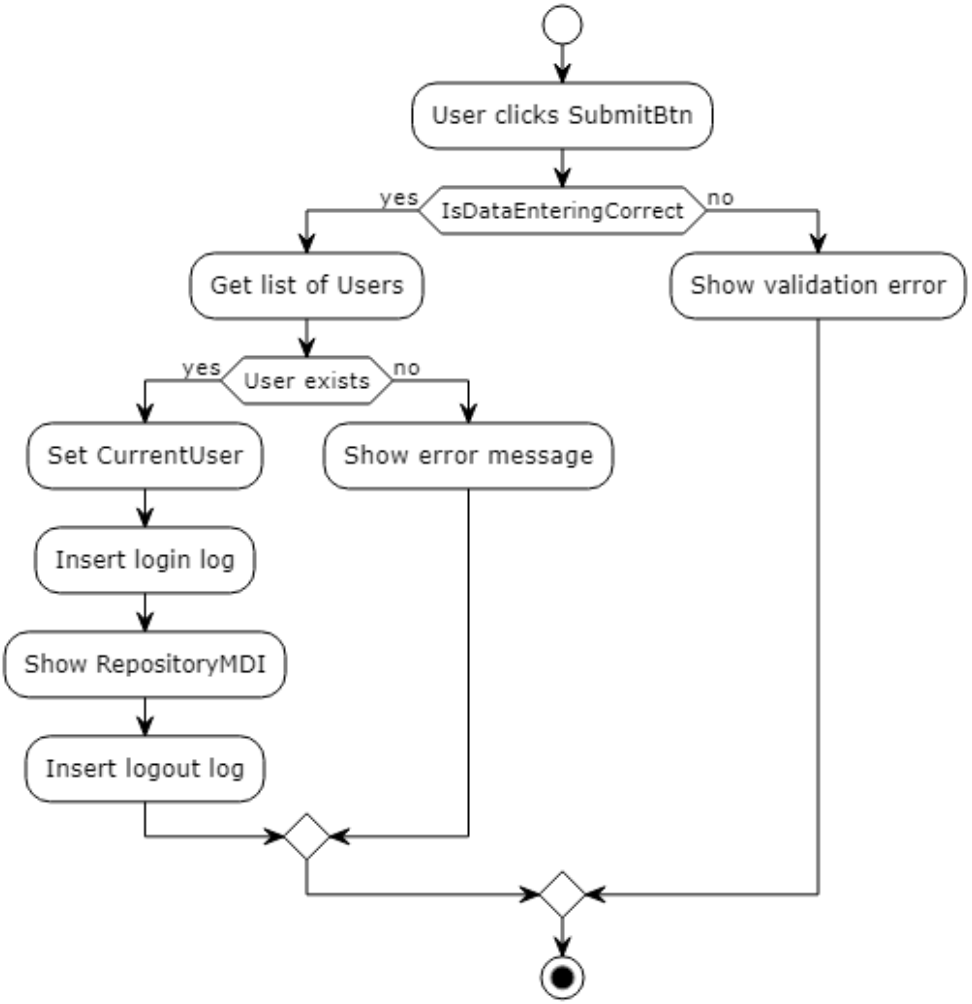

Рис. 3.3. Процес «Авторизація користувача»

На рис. 3.4 представлена діаграма діяльності, яка докладно ілюструє процедуру завантаження файлів до *Google Drive* у рамках проекту «Репозиторій». Ця діаграма важлива для ілюстрації кроків, необхідних користувачу для успішного завантаження документів у хмарне сховище. Вона показує весь процес, починаючи від вибору файла користувачем, перевірки файлу системою, підтвердження завантаження, та закінчуючи остаточними кроками системи для розміщення файлу у відповідному місці зберігання.

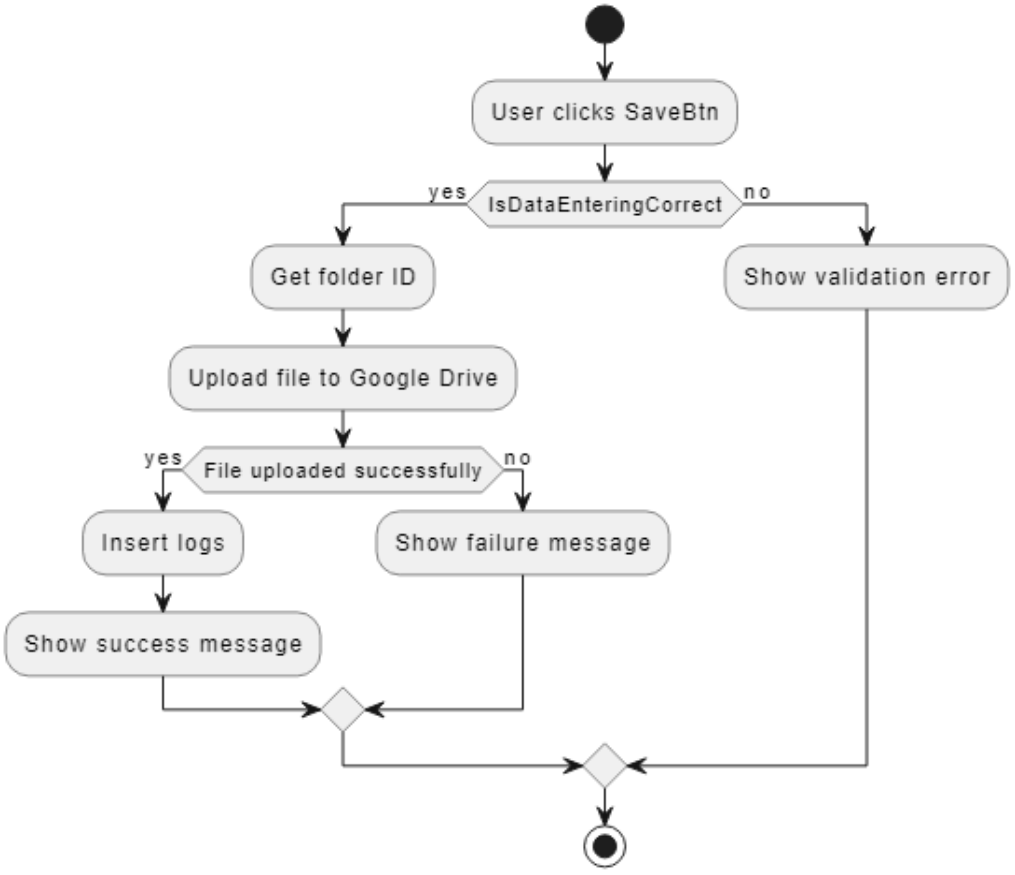

Рис. 3.4. Процес «Завантаження у репозиторій»

Процедура завантаження файлів у системі «Репозиторій» розпочинається з активації користувачем кнопки «*SaveBtn*». Спочатку система перевіряє, чи файл вибрано для завантаження. У випадку помилки, користувач отримує повідомлення та процес зупиняється. При успішній валідації, визначається цільова папка. Потім відбувається завантаження на *Google Drive*, результати фіксуються в журналі. Діаграма важлива для зрозуміння процесу та потенційного вдосконалення взаємодії з системою.

На рис. 3.5 знаходиться схема процесу, яка демонструє методику перевірки файлів, що зберігаються в репозиторії. Ця процедура починається з активації інтерфейсу «*ReviewFilesForm*», після чого система приступає до вибірки файлів з *Google Drive*. Вибрані файли відображаються у вигляді ієрархічної структури дерева, де основний рівень позначений як рівень 0.

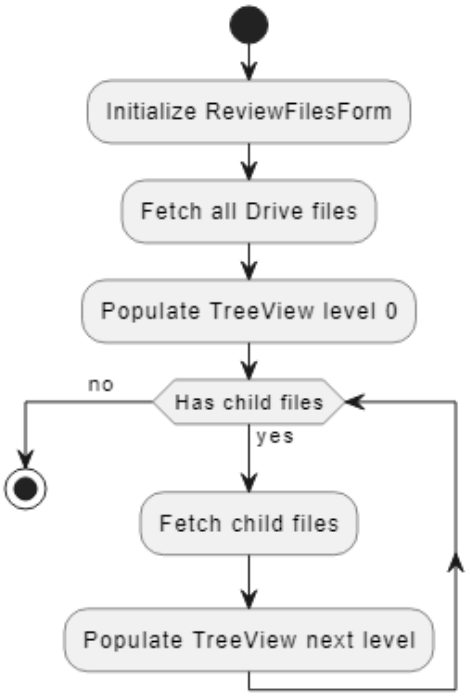

Рис. 3.5. Процес «Перевірка файлів»

У даній системі проводиться аналіз на наявність підпорядкованих елементів у файлах. Якщо такі елементи знайдені, вони додаватимуться рекурсивно до ієрархічної структури на наступні рівні. Процедура продовжує розкривати файли та їх підкатегорії, забезпечуючи всебічний огляд документації. Процес триває, поки не буде відображено всі файли та їх внутрішні структури.

На рис. 3.6 показано процес пошуку файлів. Він починається з активації інтерфейсу «*SearchFilesForm*», де користувачі можуть вказати критерії пошуку в текстове поле «*SearchTBox*». Після натискання кнопки «*SearchBtn*», система запускає метод «*GetRaportForClientFIO*», який шукає файли відповідно до заданих параметрів. Знайдені файли збираються у колекцію «*_DriveFileList*».

Наступний крок - ініціація текстового поля «*RaportTBox*», яке відображає результати пошуку. Система форматує кожен знайдений елемент з «*_DriveFileList*» в

окремий рядок, додаючи його до «*RaportTBox*» для перегляду користувачем. Пошук завершується, коли всі результати представлені на екрані.

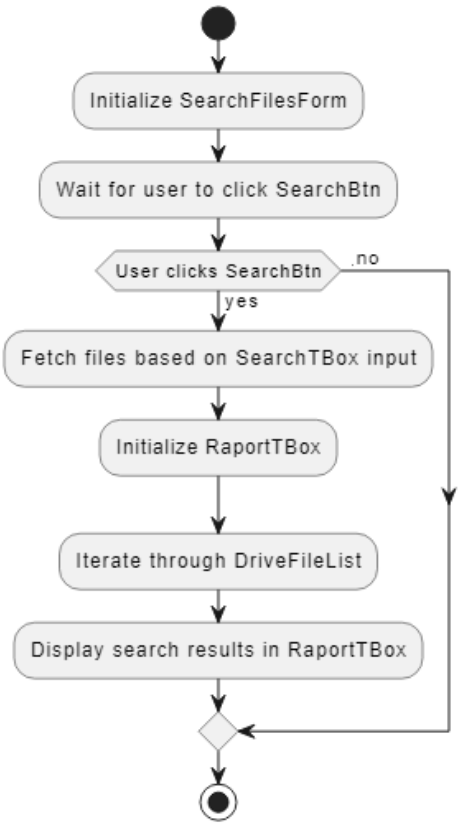

Рис. 3.6. Процес «Пошук файлів»

Після того, як були успішно розроблені та представлені діаграми діяльності, наступним кроком стало створення діаграм послідовності. Ці діаграми відіграють ключову роль у візуалізації взаємодії між компонентами системи, відображаючи їх взаємозв'язки та взаємодії у впорядкованому хронологічному порядку. Вони допомагають зрозуміти не тільки структуру та функціональність системи, але й її поведінкові аспекти в різних сценаріях використання.

Однією з важливих діаграм послідовності, яка була розроблена, є діаграма, що демонструє процес конфігурації репозиторію. Ця діаграма, представлена на рис. 3.7, в деталях показує кроки, які виконуються під час налаштування репозиторію. Через послідовне відображення кожного кроку, діаграма надає чітке уявлення про порядок дій та взаємодію між різними компонентами системи, що сприяє глибшому розумінню механізмів конфігурації репозиторію та їх взаємного впливу один на одного.

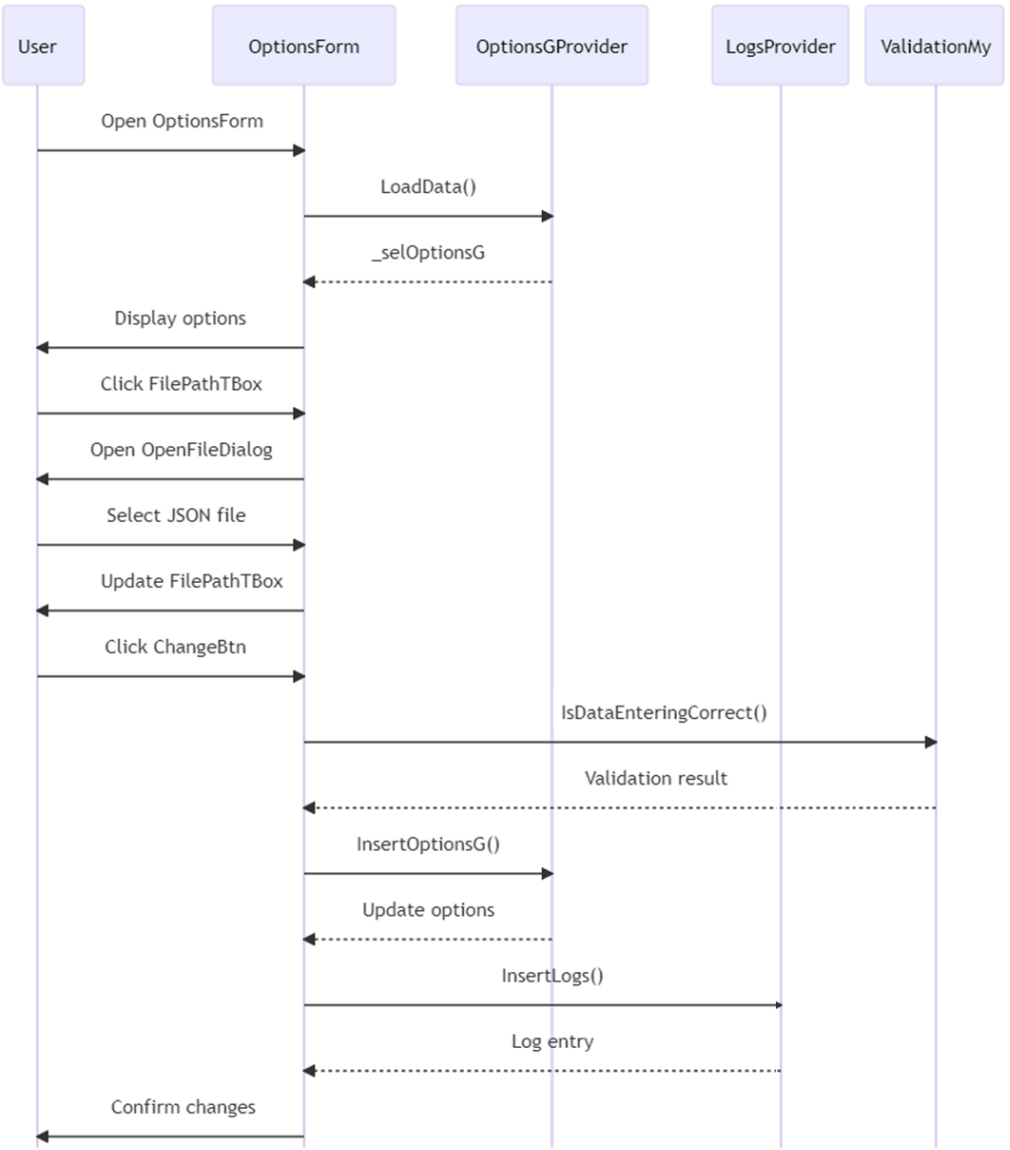

Рис. 3.7. Процес «Налаштування репозиторію»

На рис. 3.7 представлено схематичне зображення процесу налаштування доступу до *Google Drive* в рамках роботи системи. Діаграма включає в себе важливі компоненти, такі як користувач, інтерфейс налаштувань (*OptionsForm*), менеджер налаштувань (*OptionsGProvider*) та *API Google Drive*.

Процес починається з моменту, коли користувач взаємодіє з формою налаштувань, вводячи необхідні дані, як-от назву програми і шлях до файлу налаштувань. Завершивши введення, користувач ініціює збереження налаштувань, натискаючи на відповідну кнопку. Введені дані передаються з форми налаштувань до провайдера опцій, який займається їх записом у базу даних.

Коли настає час синхронізації з *Google Drive*, інтерфейс налаштувань використовує збережені параметри для авторизації в *API Google Drive*. Після успішного входу система здійснює витягування файлів з *Google Drive* та їх збереження в локальну базу даних через менеджера налаштувань. Завершується весь процес реєстрацією інформації про синхронізацію у журналі логів та відображенням сповіщення для користувача про успішне виконання налаштувань.

Рис. 3.8 демонструє діаграму, яка докладно відображає етапи процедури синхронізації локальної бази даних із зовнішнім хмарним сховищем *Google Drive*. Ця діаграма послідовності ретельно ілюструє кожен крок задіяний у цьому процесі.

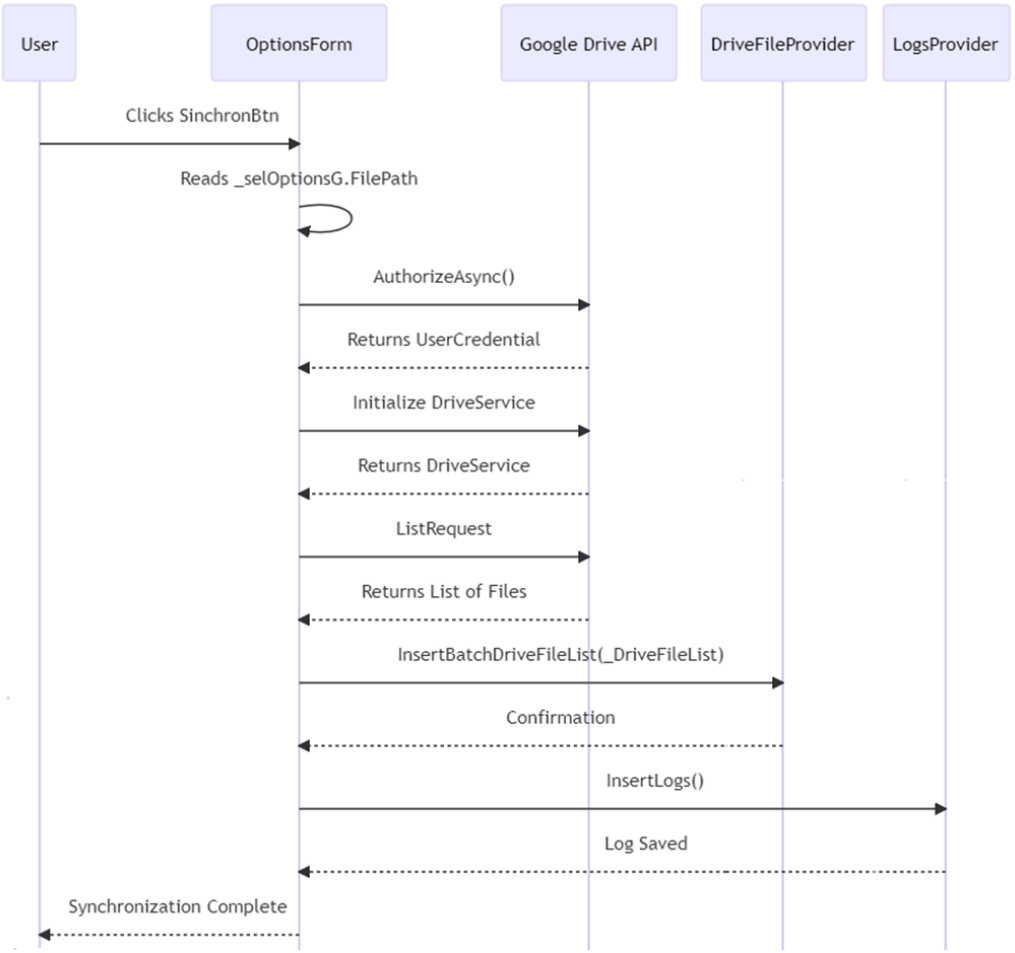

Рис. 3.8. Процес «Синхронізація бази даних із репозиторієм»

Початок процесу відбувається, коли користувач активізує функціонал за допомогою кнопки «*SinchronBtn*», що запускає подію «*Click*». Далі система реалізує авторизацію користувача через механізм *OAuth2*, використовуючи спеціальний файл із конфіденційними даними. У випадку успішної авторизації ініціюється сервіс

Google Drive API, який має доступ до репозиторію за допомогою авторизаційних даних. Наступним кроком є відправлення запиту до *Google Drive* для отримання переліку файлів. Запит включає різні параметри, такі як ліміт файлів на сторінку та конкретні поля, які потрібно витягти. Виявлені файли інтегруються в структуру віджету *TreeView*, що полегшує навігацію користувачів по каталогах і файлах.

Заключний етап полягає в синхронізації знайдених файлів із локальною базою даних. Цей крок забезпечує оновлення системи актуальною інформацією. На завершення, в системному журналі фіксується інформація про успішно здійснену синхронізацію.

Рис. 3.9 ілюструє процес створення структури каталогу документів у сервісі *Google Drive*, що є частиною проекту управління електронними документами.

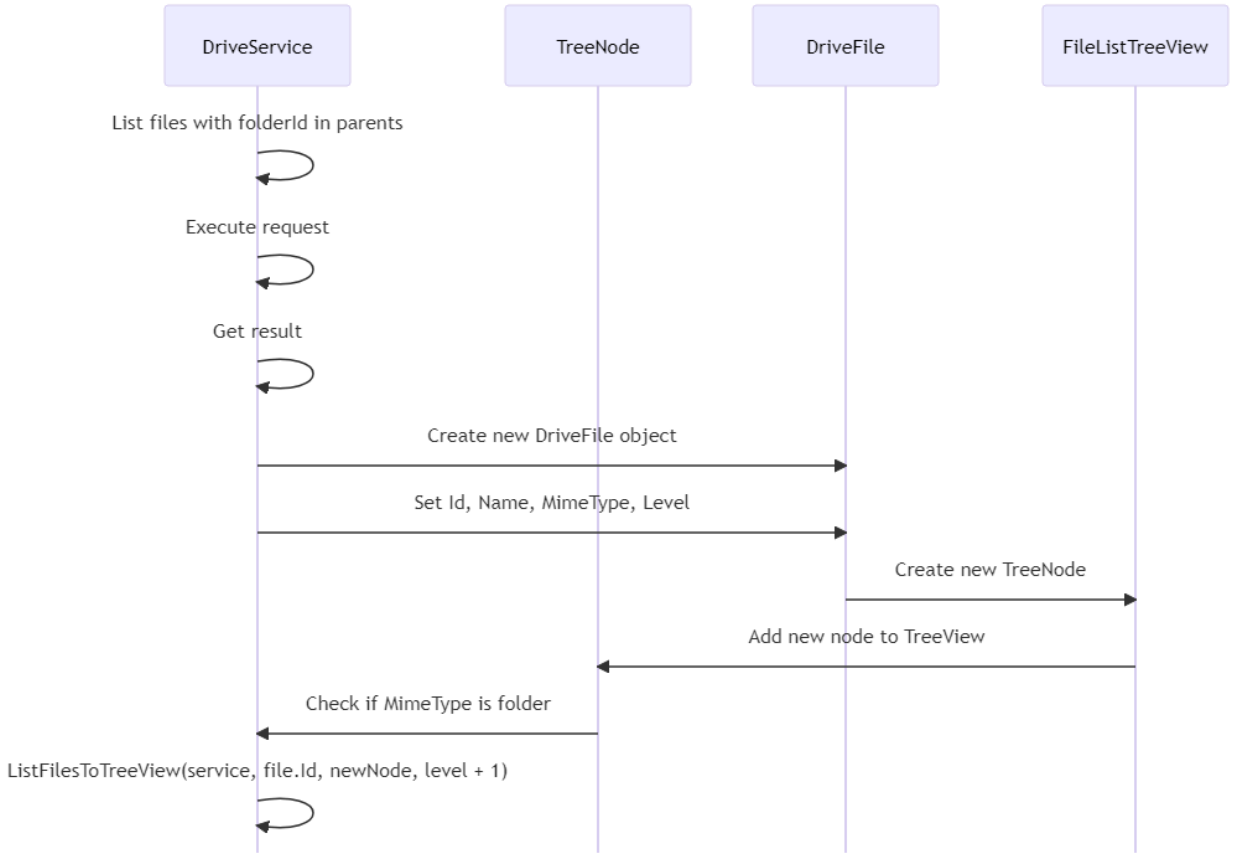

Рис. 3.9. Діаграма побудови дерева каталогу документів репозиторію

Процес активується через метод *ListFileToTreeView*. Для його виконання необхідно задати такі параметри: *Google Drive* сервіс, ідентифікатор теки, базовий вузол для дерева каталогів та рівень глибини ієрархії. Він розпочинається з відправлення запиту до *API Google Drive*, щоб отримати список файлів у визначеній

теці. Після отримання списку, програма перевіряє наявність файлів у ньому. Якщо файли існують, для кожного з них створюється відповідний вузол у структурі дерева каталогів, причому кожен вузол пов'язаний з унікальним ідентифікатором файлу. У тому випадку, коли знайдений файл є текою, метод *ListFileToTreeView* повторно викликається для цієї теки, що спричиняє зростання рівня ієрархічної вкладеності. Таким чином, на діаграмі демонструється рекурсивна логіка формування всієї структури дерева каталогів.

3.4. Висновки до розділу

У даному розділі було здійснено глибокий аналіз та детальне проектування системи електронного документообігу, що включає в себе важливі елементи. Спершу відбулося формулювання вимог до системи, включаючи як функціональні, так і нефункціональні аспекти, що допомогло зрозуміти очікування та стандарти продуктивності. Було визначено акторів системи та їх цілі, що забезпечило відповідність функціональності системи до потреб користувачів. Аналіз варіантів використання системи виявив основні взаємодії між користувачами та системою, сприяючи її ефективності.

Були створені діаграми прецедентів (*use-case*), які відіграли ключову роль у візуалізації інтеракцій між акторами та системою, надаючи чітке уявлення про процеси взаємодії. Розроблені діаграми бізнес-процесів, такі як діаграми діяльності для основних процесів (авторизація користувача, завантаження файлів у репозиторій, перегляд та пошук файлів), детально відобразили робочі процеси в системі. Діаграми послідовностей для налаштування репозиторію та синхронізації бази даних з репозиторієм надали додаткове розуміння про взаємодію компонентів системи.

Виконана робота в рамках цього розділу заклала міцний підґрунтя для дальшого розвитку системи електронного документообігу. Уважне визначення вимог, розробка діаграм прецедентів і аналіз бізнес-процесів забезпечили чіткий

концептуальний план системи, відповідного високим стандартам доступності, безпеки та зручності для кінцевих користувачів.

РОЗДІЛ 4
РОЗРОБКА ПРОГРАМНОЇ СИСТЕМИ УПРАВЛІННЯ ТА РЕЗЕРВНОГО КОПІЮВАННЯ

4.1. Проєктування та розробка бази даних

В системах управління документацією база даних є ключовим компонентом. Вона служить не тільки як сховище для документів і їх метаданих, але й забезпечує можливості для продуктивного пошуку, фільтрування та сортування даних. Розробка бази даних потребує ґрунтовного аналізу та стратегічного планування, включаючи визначення структури даних, встановлення їх взаємозв'язків, а також розробку методів їх зберігання і доступу.

При створенні репозиторію особлива увага зосереджується на розробці ефективної та безпечної структури бази даних, щоб забезпечити високу продуктивність і надійність системи. Основні таблиці бази даних включають:

– таблицю «*Users*», яка містить основні дані про користувачів системи, а саме: ідентифікатор користувача (*UsersId*), ім'я (*FirstName*), прізвище (*LastName*), логін для входу в систему (*UsersName*), зашифрований пароль (*UsersPassword*), ідентифікатор ролі у системі (*RoleId*), додатковий опис (*Description*) та електронну адресу (*Email*);

– таблиця «*Logs*» використовується для зберігання інформації про активність користувачів, включаючи нумератор записів (*LogsId*), ідентифікатор користувача, що ініціював подію (*UsersId*), опис події (*EventNameShow*) та точний час та дата відбування події (*EventDate*);

– таблиця «*DriveFile*» призначена для зберіганні даних файлів у репозиторії, включаючи унікальний номер файлу в системі (*DriveFileId*), ідентифікатор файлу в репозиторії (*Id*), повну назву файлу (*Name*), тип файлу (*MimeType*) та рівень доступу до файлу (*Level*);

– таблиця «*OptionsG*» містить глобальні налаштування системи, такі як ідентифікатор налаштувань (*OptionsGId*), шлях до файлу налаштувань (*FilePath*) та назву програми для роботи з файлами в репозиторії (*ApplicationName*).

Ці таблиці створюють згуртовану структуру даних, розроблену з метою ефективного архівування, організації та обробки документів у межах системи управління електронними документами. Така структура є ключовою для забезпечення доступності документів, управління їх версіями та контролю доступу користувачів.

На рис. 4.1 візуально ілюстрована фізична модель бази даних, в якій представлені основні компоненти та їх взаємозв'язки. Це забезпечує повне розуміння структури та організації даних у репозиторії електронних документів.

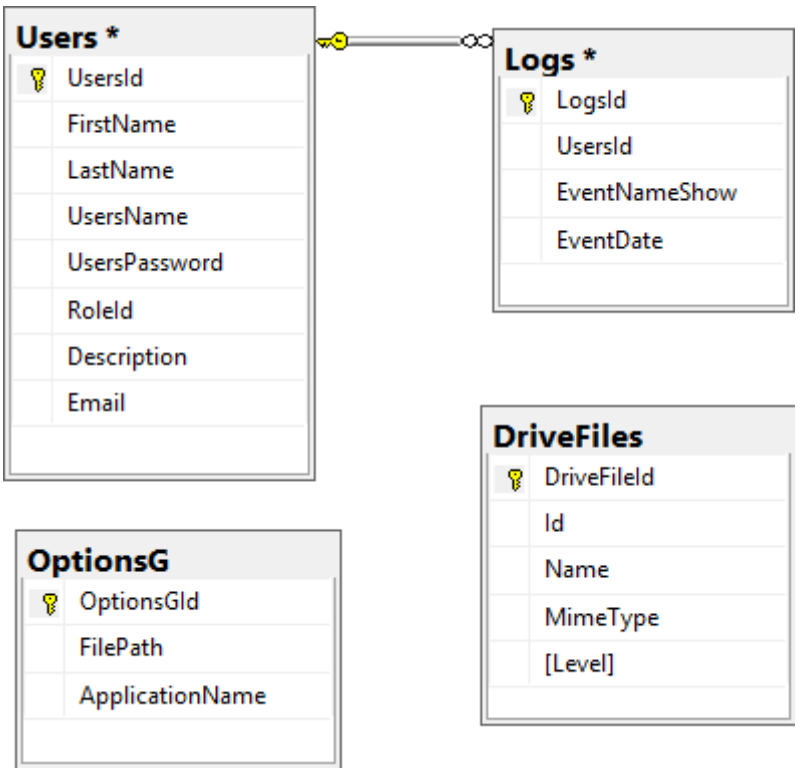

Рис. 4.1. Модель бази даних

В контексті цієї роботи, кодування структур даних та реалізація обмежень для забезпечення цілісності даних є важливими. Цей етап включає розробку набору *SQL*-скриптів для створення та ініціації таблиць, визначення їх взаємодій, розробку індексів для покращення ефективності пошуку, а також створення тригерів для

автоматизації деяких процесів. Повний перелік цих *SQL*-скриптів можна знайти в додатку А до цього наукового дослідження.

4.2. Розробка архітектури системи

Архітектура програмного застосунку є основою, яка формує організаційну структуру програми, включаючи як внутрішні, так і зовнішні взаємозв'язки між її компонентами. Вона встановлює не лише технічну реалізацію системи, а й методики її розвитку та підтримки. Для даної системи було вибрано архітектуру з трьома рівнями, що обґрунтовано з кількох позицій:

– модульність та адаптивність. Така архітектура дозволяє чітко розділити функції системи на різні рівні, сприяючи легшості управління та можливості адаптації;

– легкість обслуговування та розширення. Окремі рівні можуть оновлюватися або модифікуватися незалежно один від одного, забезпечуючи стабільність системи при внесенні змін;

– оптимізація продуктивності та масштабованість. Розподіл функціоналу між рівнями дозволяє ефективніше використовувати ресурси та оптимізувати загальну продуктивність системи;

– забезпечення безпеки інформації. Різні рівні можуть мати окремі механізми безпеки, що підвищує загальний рівень захисту даних;

– спрощення розробки та тестування. Незалежне тестування кожного рівня упрощує ці процеси, сприяючи поліпшенню якості загального продукту.

Перелічені аспекти роблять дану архітектуру придатною для системи зберігання електронних документів, забезпечуючи баланс між адаптивністю, оптимізацією та безпекою.

Трирівнева архітектура представляє собою структурний метод в розробці програмного забезпечення, розбиваючи функціонал на три різні рівні:

– презентаційний рівень. Зосереджений на взаємодії користувача з системою, включаючи розробку користувацького інтерфейсу та надання доступу до функцій системи через десктопний застосунок;

– логічний рівень або рівень бізнес-логіки. Являє собою серцевину системи, обробляючи запити, виконуючи операції та координуючи передачу даних між презентаційним та даними рівнями;

– рівень даних. Займається зберіганням документів та метаданих, а також виконанням операцій *CRUD* для взаємодії з логічним рівнем.

На рис. 4.2 представлена загальна структура розробленої системи.

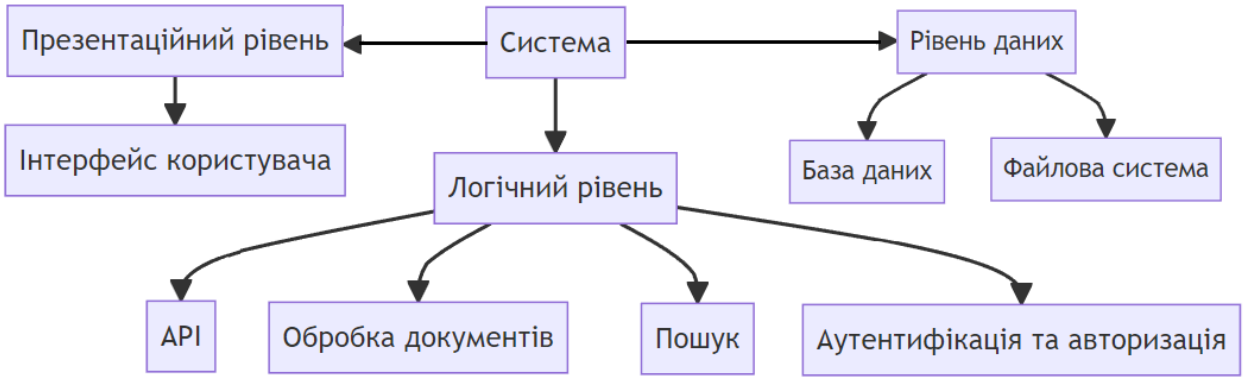

Рис. 4.2. Архітектура системи електронного репозиторію

Діаграма ілюструє трирівневу структуру ІС, що складається з трьох основних компонентів: презентаційного, логічного та даних рівнів. «Система» розташована в центрі діаграми та має прямі зв'язки з кожним з цих рівнів.

Презентаційний рівень слугує для взаємодії користувачів з системою через графічний інтерфейс. Логічний рівень включає *API* для комунікації з іншими елементами системи, а також модулі для обробки та пошуку документів, ідентифікації користувачів і управління доступом. Рівень даних відповідає за зберігання і управління даними, що охоплює базу даних для структурованих даних і файлову систему для неструктурованих чи об'ємних даних. Кожен з цих елементів має свою специфічну роль, сприяючи гнучкості, масштабуванню та надійності системи.

Під час розробки системи «Репозиторій» особлива увага була зосереджена на створенні шару доступу до даних, який функціонує як посередник між базою даних і логічним рівнем, забезпечуючи гнучке управління документами та метаданими. Для цього були розроблені чотири спеціалізовані класи, кожен з яких спеціалізується на взаємодії з конкретною таблицею в базі даних.

На діаграмі класів (рис. 4.3) чітко відображена архітектура шару доступу до даних, з детальним описом методів та атрибутів кожного класу. Це має важливе значення для розуміння структури системи та її подальшого розвитку.

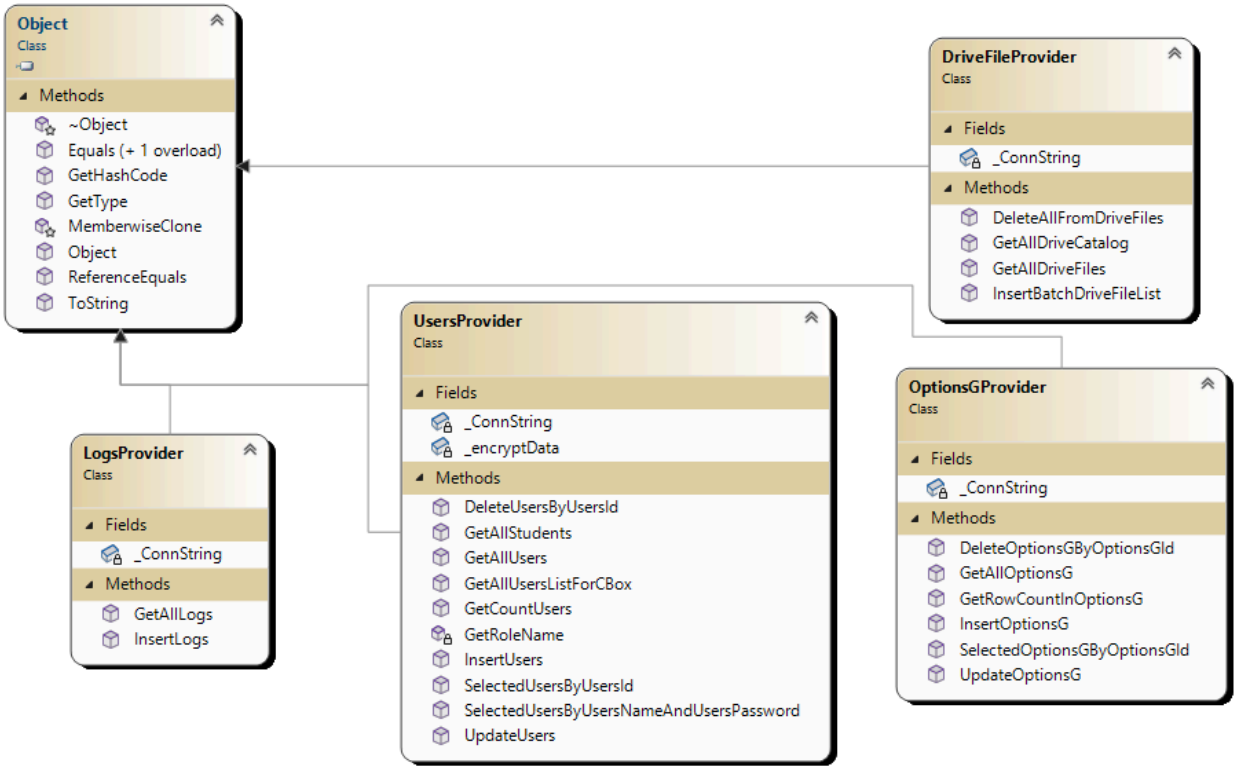

Рис. 4.3. Діаграма класів для рівня даних

На діаграмі класів, що представляє рівень даних системи, чітко виділені чотири ключові класи, кожен з яких відповідає за свої унікальні функції:

– клас «*LogsProvider*». Цей клас виконує важливу роль у фіксації різноманітних дій і подій, що відбуваються в системі. Він відстежує такі дії, як завантаження файлів, зміни в конфігураціях, а також фіксує як успішні, так і невдалі спроби доступу до системи. Цей клас є невід'ємною частиною системи з точки зору аудиту діяльності користувачів та забезпечення безпеки системи;

– клас «*OptionsProvider*». Зосереджений на управлінні та конфігурації системи. Він включає параметри безпеки, налаштування зберігання файлів, правила доступу, та інші важливі конфігураційні аспекти. Централізований підхід до управління цими налаштуваннями через один клас робить процес конфігурації більш ефективним і зручним;

– клас «*DriveFileProvider*». Спеціалізується на обробці файлів, що зберігаються в репозиторії. Він включає в себе методи для завантаження, збереження, оновлення, видалення файлів та доступу до їх метаданих. Цей клас грає ключову роль у забезпеченні цілісності та доступності файлів у системі, що є критично важливим для зберігання та управління документацією;

– клас «*UsersProvider*». Фокусується на управлінні обліковими записами користувачів. Він об'єднує функції, пов'язані з аутентифікацією та авторизацією, створенням нових облікових записів, відновленням паролів, та іншими діями, що стосуються керування користувацькими профілями. Це забезпечує важливий елемент управління доступом та безпеки в системі.

Під час створення системи особлива увага була зосереджена на розробці користувацького інтерфейсу, який є важливим аспектом для забезпечення зручності та ефективності взаємодії користувачів із системою. Розроблений інтерфейс включає в себе низку різноманітних форм, кожна з яких обладнана унікальним набором елементів управління, таких як кнопки, текстові поля, вибіркові списки та таблиці. Ці інтерактивні компоненти дозволяють користувачам здійснювати безпосередню взаємодію з різними сегментами системи, використовуючи спеціалізовані обробники подій для керування діями системи.

Користувацький інтерфейс забезпечує широкі можливості для користувачів, від простого перегляду наявних документів до введення нових даних, реалізації запитів пошуку та отримання результатів в зручному для користувача форматі. Такий інтерфейс сприяє інтуїтивному розумінню та продуктивному використанню системи, що полегшує адаптацію користувачів та підвищує їхню задоволеність від роботи з системою.

Для глибшого вивчення архітектури користувацького інтерфейсу була створена спеціальна діаграма, представлена на рис. 4.4. Ця діаграма наглядно ілюструє розміщення різних форм і елементів управління, відображаючи їх взаємозв'язок та взаємодію з іншими компонентами системи, що допомагає зрозуміти загальну логіку роботи та взаємодії внутрішніх частин системи.

Рис. 4.4. Діаграма класів для *UI* - рівня

На *UI*-рівні діаграми представлено десять класів, які є розширеннями базового класу *Form* і мають візуальні інтерфейси. Кожен із цих класів виконує певну функцію у системі:

– клас «*RepositoryMDI*» функціонує як головне вікно інтерфейсу, координуючи взаємодію користувачів з різними формами та елементами управління, надаючи доступ до широкого спектру функцій системи;

– клас «*ReviewFilesForm*» дозволяє переглядати перелік доступних документів та відкривати їх для детального перегляду, створюючи інтерфейс для легкої навігації по репозиторію;

– клас «*UploadFileForm*» пропонує інтерфейс для завантаження нових документів із засобами вибору файлів та вводу метаданих для них;

– клас «*SearchFilesForm*» уможливлює пошук документів у репозиторії, дозволяючи користувачам задавати критерії пошуку та отримувати відфільтрований список документів;

– клас «*LoginForm*» відповідальний за процес аутентифікації користувачів, забезпечуючи поля для вводу логіна та пароля та їх подальшу валідацію;

– клас «*OptionsForm*» призначений для налаштувань конфігурації системи, включаючи параметри безпеки, управління зберіганням файлів та інші важливі налаштування;

– клас «*PersonalizationForm*» дозволяє користувачам налаштовувати інтерфейс системи відповідно до їх переваг, включаючи теми дизайну та інші персоналізовані опції;

– клас «*SystemLogForm*» є інструментом для перегляду системних журналів, що є важливим для здійснення аудиту та моніторингу безпеки;

– клас «*UpdateUsersForm*» надає функціональність для оновлення інформації про користувачів, зміни їхніх ролей та прав доступу;

– клас «*UsersForm*» використовується для управління списком користувачів, що включає функції перегляду, створення та видалення користувацьких облікових записів, що є ключовим для адміністрації системи.

В архітектурі програми об'єкти різних класів ініціюються та активуються у відповідь на дії користувачів в графічному інтерфейсі. Останній крок у розробці архітектури полягає у формуванні бізнес-логіки додатку, яка відіграє вирішальну роль, оскільки саме на цьому етапі визначаються ключові правила і алгоритми для операцій, таких як завантаження, пошук, оновлення та видалення документів з репозиторію.

Розроблення бізнес-логіки з розділенням на окремі модулі сприяє збільшенню модульності та гнучкості системи, роблячи її легшою у підтримці та масштабуванні. Це також дозволяє вносити зміни або оновлення до окремих частин системи без впливу на її інші компоненти, що є важливим фактором для її стабільності та розвитку.

Розділення бізнес-логіки від рівня даних та користувацького інтерфейсу надає системі додаткову адаптивність, дозволяючи змінювати бази даних або інтерфейси, не переробляючи основну логіку додатку. Ця гнучкість є особливо важливою в сучасному швидкозмінному технологічному середовищі, дозволяючи системі ефективно адаптуватися до нових бізнес-вимог та технологічних трендів.

Детальну структуру шару бізнес-логіки демонструє діаграма класів, яку можна переглянути на рис. 4.5, де відображено всі ключові компоненти та їх взаємодії в рамках системи.

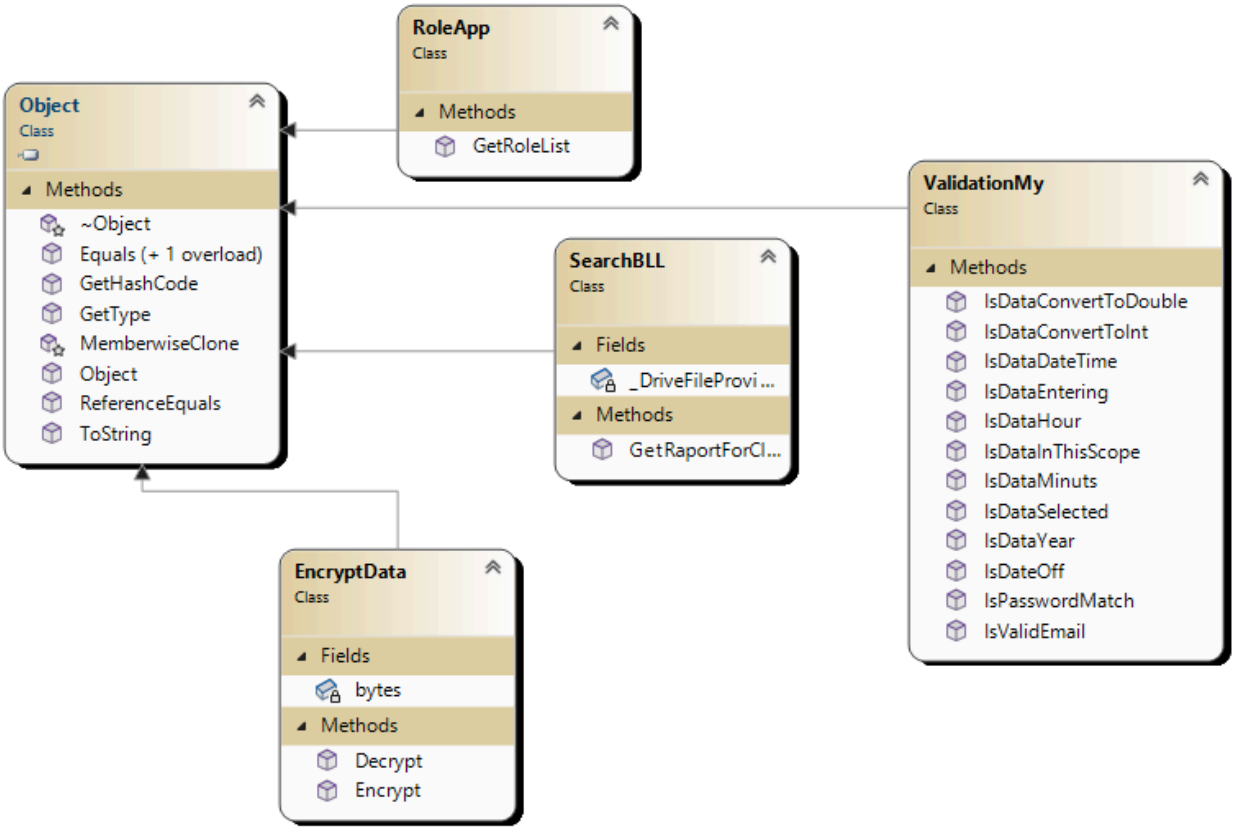

Рис. 4.5. Діаграма класів бізнес-логіки системи

На діаграмі, що відображає бізнес-логіку додатку, виокремлено чотири основні класи, які виконують специфічні задачі:

– клас «*SearchBLL*» зосереджений на реалізації пошукових функцій у репозиторії. Він забезпечує виконання складних запитів, базуючись на критеріях, заданих користувачем, які можуть включати текстові запити, фільтрацію за датою, типом файлу, автором та іншими параметрами;

– клас «*EncryptData*» відповідає за процеси шифрування та дешифрування даних, забезпечуючи їх конфіденційність та цілісність. Клас використовує передові алгоритми та ключі шифрування для забезпечення високого рівня безпеки;

– клас «*ValidationMy*» спеціалізується на валідації вхідних даних та запитів від користувачів. Це включає перевірку форматів файлів, правильності тексту у пошукових запитах, а також вірогідності користувацьких даних;

– клас «*RoleApp*» займається управлінням системою ролей та дозволів. Цей клас дозволяє динамічно регулювати доступ до різних розділів системи в залежності від ролі користувача, надаючи різні рівні доступу, наприклад, адміністраторам та звичайним користувачам.

Кожен з цих класів грає ключову роль у бізнес-логіці системи, інтегруючись з рівнем даних та користувацьким інтерфейсом для забезпечення цілісності, безпеки та високої продуктивності додатку. Розробка цих класів базується на глибокому аналізі бізнес-потреб і технологічних можливостей, що надає системі необхідну адаптивність та масштабованість.

4.3. Інтеграція з хмарним сховищем

Можливість зберігати та отримувати доступ до даних з будь-якої точки світу перетворилася з зручності у необхідність. Відходячи від практики зберігання всіх документів на локальних серверах, перехід до використання хмарних сховищ, які пропонують швидку синхронізацію та безпечне зберігання даних, стає все більш привабливим для бізнесу та освітніх організацій. Інтеграція електронних документів з хмарними сервісами, такими як *Google Drive*, є важливим етапом у розвитку цього проекту, маючи на меті розширення його функціональності, підвищення доступності та полегшення використання.

У процесі інтеграції з *Google Drive* у рамках проекту, розробленого в *Visual Studio 2022*, був використаний *NuGet*-пакет *Google.Apis.Drive.v3*. Процес додавання цього пакету включає кілька кроків:

– відкриття *NuGet Package Manager*. Це початковий крок, який полягає у відкритті менеджера пакетів *NuGet* у контексті проекту;

– пошук пакету. У вікні «*NuGet - Solution*» на вкладці «*Browse*» виконується пошук, вводячи «*Google.Apis.Drive.v3*» у поле пошуку та натискаючи *Enter*;

– встановлення пакету. Вибір пакету *Google.Apis.Drive.v3* зі списку та натискання на кнопку «*Install*», яка з'явиться поряд з назвою пакету;

– підтвердження залежностей. Відобразиться список залежностей, що встановлюються разом із *Google.Apis.Drive.v3*. Після перевірки, потрібно натиснути «*OK*»;

– ліцензійна згода. У вікні ліцензійної угоди прийняти умови, натиснувши «*I Accept*»;

– перевірка встановлення. Після завершення процесу встановлення, перейти до вкладки «*Installed*» і переконатися, що *Google.Apis.Drive.v3* з'явився у списку встановлених пакетів.

Завершення наведених вище кроків успішно інтегрувало *NuGet*-пакет *Google.Apis.Drive.v3* в проект, що підготувало основу для подальшого взаємодії з *API Google Drive*. Цей процес інтеграції відображено на рис. 4.6, де представлено конфігураційний файл пакету.

```
<?xml version="1.0" encoding="utf-8"?>
<packages>
  <package id="Google.Apis" version="1.62.0" targetFramework="net472" />
  <package id="Google.Apis.Auth" version="1.62.0" targetFramework="net472" />
  <package id="Google.Apis.Core" version="1.62.0" targetFramework="net472" />
  <package id="Google.Apis.Drive.v3" version="1.62.0.3155" targetFramework="net472" />
  <package id="Newtonsoft.Json" version="13.0.3" targetFramework="net472" />
</packages>
```

Рис. 4.6. Конфігурація файлу пакету

У контексті системи «Репозиторій», для взаємодії з *Google Drive* була розроблена спеціалізована форма «*OptionsForm*». Ця форма використовується для

налаштування параметрів *Google Drive*. При запуску форми, активується функція *LoadData*, яка перевіряє наявність налаштувань у базі даних та завантажує їх, або, у випадку їх відсутності, створює новий запис зі стандартними налаштуваннями. Процес створення нових записів налаштувань представлено на рис. 4.7.

```
private void LoadData() {
  if (_OptionsGProvider.GetRowCountInOptionsG() > 0) {
    _selOptionsG = _OptionsGProvider.SelectedOptionsGByOptionsGId(1);
  } else {
    _selOptionsG.ApplicationName = "RDSU 90542";
    _selOptionsG.FilePath =
      "client_secret_1030940971022-u5pocgdcrvlor4gotrrlla80tnutkje5.apps.googleusercontent.com.json";
    _OptionsGProvider.InsertOptionsG(_selOptionsG.ApplicationName, _selOptionsG.FilePath);
  }
  ApplicationNameTBox.Text = _selOptionsG.ApplicationName;
  FilePathTBox.Text = _selOptionsG.FilePath;
}
```

Рис. 4.7. Налаштування файлу пакету

Форма «*OptionsForm*» також містить текстове поле «*FilePathTBox*» для визначення шляху до *JSON*-файлу налаштувань. При взаємодії з цим полем, користувачу відкривається діалогове вікно, яке дозволяє обрати відповідний файл, що демонструється на рис. 4.8.

```
private void FilePathTBox_MouseClick(object sender, MouseEventArgs e) {
  using (OpenFileDialog openFileDialog = new OpenFileDialog()) {
    openFileDialog.Filter = "JSON files (*.json)|*.json";
    openFileDialog.Title = "Select a JSON file";
    if (openFileDialog.ShowDialog() == DialogResult.OK) {
      FilePathTBox.Text = openFileDialog.FileName;
    }
  }
}
```

Рис. 4.8. Метод вибору конфігураційного файлу

У рамках роботи з формою «*OptionsForm*» однією з ключових функцій є можливість ініціювати процес синхронізації з *Google Drive* за допомогою кнопки «*SinchronBtn*». Цей процес охоплює кроки авторизації користувача у *Google Drive* та завантаження переліку файлів з хмарного сховища, як це показано на рис. 4.9.

```
private void SinchronBtn_Click(object sender, EventArgs e) {
  UserCredential credential;
  using (var stream = new FileStream(_selOptionsG.FilePath, FileMode.Open, FileAccess.Read)) {
    string credPath = "token.json";
    credential = GoogleWebAuthorizationBroker.AuthorizeAsync(
        GoogleClientSecrets.Load(stream).Secrets,
        Scopes,
        "user",
        CancellationToken.None,
        new FileDataStore(credPath, true)).Result;
    Console.WriteLine("Credential file saved to: " + credPath);
  }
  // Create Drive API service.
  var service = new DriveService(new BaseClientService.Initializer() {
    HttpClientInitializer = credential,
    ApplicationName = _selOptionsG.ApplicationName,
  });
  // Define parameters of request.
  FilesResource.ListRequest listRequest = service.Files.List();
  listRequest.PageSize = 10;
  listRequest.Fields = "nextPageToken, files(id, name)";

  // List files.
  //ListFiles(service, "root", 0);
  ListFilesToTreeView(service, "root", null, 0);
  _DriveFileProvider.InsertBatchDriveFileList(_DriveFileList);
  _LogsProvider.InsertLogs(LoginForm.CurrentUser.UsersId,
    "Були синхронізовано інформацію про наявність документів", DateTime.Now);
}
```

Рис. 4.9. Процедура синхронізації з «*Google Drive*»

В межах проекту «Репозиторій» було розроблено форму «*ReviewFilesForm*», призначену для відображення файлів, що зберігаються в *Google Drive*. При її відкритті активізується конструктор класу «*ReviewFilesForm*», який викликає метод «*InitializeComponent*» для ініціації компонентів форми, відображено на рис. 4.10.

```
public ReviewFilesForm() {
  InitializeComponent();
  _DriveFileList = _DriveFileProvider.GetAllDriveFiles();
  PopulateTreeView();
}
```

Рис. 4.10. Ініціалізація форми «*ReviewFilesForm*»

У конструкторі класу реалізовано вилучення списку файлів із *Google Drive* за допомогою класу «*DriveFileProvider*». Отриманий список файлів тимчасово зберігається у приватній змінній «*_DriveFileList*». Для візуалізації файлів та папок з хмарного сховища у вигляді деревоподібної структури, де кожна папка може розгортатися для відображення свого вмісту, було розроблено метод «*PopulateTreeView*». Деталі цього методу можна переглянути на рис. 4.11.

```
private void PopulateTreeView() {
    FileListTreeView.Nodes.Clear(); // Очищуємо TreeView
    Dictionary<int, TreeNode> lastNodesAtEachLevel = new Dictionary<int, TreeNode>();
    foreach (DriveFile file in _DriveFileList) {
        TreeNode newNode = new TreeNode(file.Name) { Tag = file.Id };
        // Якщо це кореневий рівень, додаємо безпосередньо до TreeView
        if (file.Level == 0) {
            FileListTreeView.Nodes.Add(newNode);
            lastNodesAtEachLevel[file.Level] = newNode; // Запам'ятовуємо останній вузол на рівні
        } else if (lastNodesAtEachLevel.ContainsKey(file.Level - 1)) {
            // Якщо це не кореневий рівень, додаємо як дочірній до останнього вузла попереднього рівня
            TreeNode parentNode = lastNodesAtEachLevel[file.Level - 1];
            parentNode.Nodes.Add(newNode);
            lastNodesAtEachLevel[file.Level] = newNode; // Оновлюємо останній вузол на поточному рівні
        }
        // Якщо це папка, то очищаємо записи для вищих рівнів
        if (file.MimeType == "application/vnd.google-apps.folder") {
            // Видаляємо всі вузли, які є вищими за поточний рівень
            var keysToRemove = lastNodesAtEachLevel.Keys.Where(k => k > file.Level).ToList();
            foreach (var key in keysToRemove) {
                lastNodesAtEachLevel.Remove(key);
            }
        }
    }
}
```

Рис. 4.11. Реалізація методу «*PopulateTreeView*»

Метод *PopulateTreeView* відіграє ключову роль у організації та демонстрації файлів з хмарного сховища у форматі деревоподібної структури. Він починає свою роботу з очищення існуючих вузлів у компоненті *TreeView*, щоб уникнути повторного виведення файлів. Для кожного елемента зі списку *_DriveFileList* метод створює новий вузол, який включає назву та ідентифікатор файлу. Ці вузли потім додаються до *TreeView*: якщо файл належить до кореневого рівня, він вставляється безпосередньо в *TreeView*; у випадку, коли файл є частиною підкатегорії, він розміщується як дочірній елемент відповідного вищого рівня.

Коли файл являє собою папку, метод видаляє усі надрівневі вузли в *TreeView*, забезпечуючи коректне та точне відображення структури папок та файлів. Це робить навігацію по файловій системі хмарного сховища зрозумілою та зручною для користувачів через графічний інтерфейс додатку.

У контексті проекту «Репозиторій» була розроблена форма «*UploadFileForm*» для завантаження файлів у хмарне сховище *Google Drive*. На початковому етапі конструктора цієї форми виконується ініціалізація компонентів за допомогою методу *InitializeComponent*, процес якого відображений на рис. 4.12.

```
public UploadFileForm() {
    InitializeComponent();
    LoadAllDate();
}
```

Рис. 4.12. Створення конструктору форми

Під час ініціалізації форми в конструкторі виконується метод «*LoadAllDate*», який інтегрує та обробляє різноманітні дані – налаштування системи, файлову структуру та інформацію про користувачів – для їх ефективного відображення та управління в різних компонентах інтерфейсу. Візуалізація цього процесу представлена на рис. 4.13.

```
private void LoadAllDate() {
    _selOptionsG = _OptionsGProvider.SelectedOptionsGByOptionsGId(1);
    _DriveFileList = _DriveFileProvider.GetAllDriveCatalog();
    FoldersCBox.DataSource = _DriveFileList;
    FoldersCBox.ValueMember = "DriveFileId";
    FoldersCBox.DisplayMember = "Name";
    _UsersL = _UsersProvider.GetAllUsers();
    LoadDataInGridView(_UsersL);
}
```

Рис. 4.13. Реалізація методу «*LoadAllDate*»

Метод «*LoadAllDate*» виконує декілька ключових завдань для завантаження та відображення даних у формі. На першому етапі, метод витягує налаштування системи з відповідного провайдера налаштувань, використовуючи заданий ідентифікатор (наприклад, 1), і зберігає ці налаштування у приватну змінну «*_selOptionsG*». Далі, метод звертається до провайдера файлів *Drive* для отримання списку всіх файлів і каталогів з *Google Drive*, зберігаючи цей список у змінній «*_DriveFileList*».

Наступним кроком є встановлення отриманого списку файлів як джерела даних для випадаючого списку (*FoldersCBox*), використовуючи для цього властивості «*ValueMember*» та «*DisplayMember*», які вказують на ідентифікатор та назву файлу відповідно. Також метод забезпечує завантаження та відображення списку користувачів, отриманих від провайдера користувачів (*_UsersProvider*), використовуючи цей список для заповнення таблиці або сітки даних за допомогою методу «*LoadDataInGridView*».

Ця інтеграція та обробка даних дозволяє забезпечити ефективне та зручне управління інформацією в рамках форми, роблячи інтерфейс більш інтуїтивно зрозумілим та функціональним для користувачів.

```
private void FilePathTBox_MouseClick(object sender, MouseEventArgs e) {
  // Створюємо новий екземпляр OpenFileDialog
  OpenFileDialog openFileDialog = new OpenFileDialog();
  // Встановлюємо фільтр типів файлів, які можна вибирати
  openFileDialog.Filter = "All Files (*.*)|*.*";
  // Відкриваємо діалогове вікно
  if (openFileDialog.ShowDialog() == DialogResult.OK) {
    // Отримуємо та встановлюємо шлях до вибраного файлу
    string filePath = openFileDialog.FileName;
    FilePathTBox.Text = filePath;
    // Виділення назви файлу з розширенням
    _FileName = Path.GetFileName(filePath);
  }
}
```

Рис. 4.14. Код методу «*FilePathTBox_MouseClick*»

Функція вибору та завантаження файлів користувачами до хмарного сховища реалізована через подію «*FilePathTBox_MouseClick*», яка активується при кліку на текстове поле «*FilePathTBox*», як це зображено на рис. 4.15.

```
private void FilePathTBox_MouseClick(object sender, MouseEventArgs e) {
  // Створюємо новий екземпляр OpenFileDialog
  OpenFileDialog openFileDialog = new OpenFileDialog();
  // Встановлюємо фільтр типів файлів, які можна вибирати
  openFileDialog.Filter = "All Files (*.*)|*.*";
  // Відкриваємо діалогове вікно
  if (openFileDialog.ShowDialog() == DialogResult.OK) {
    // Отримуємо та встановлюємо шлях до вибраного файлу
    string filePath = openFileDialog.FileName;
    FilePathTBox.Text = filePath;
    // Виділення назви файлу з розширенням
    _FileName = Path.GetFileName(filePath);
  }
}
```

Рис. 4.15. Реалізація події «*FilePathTBox_MouseClick*»

При виклику цієї події спочатку створюється екземпляр класу «*OpenFileDialog*», стандартного елементу в *Windows* для відкриття діалогових вікон вибору файлів. Для цього діалогового вікна задається фільтр, що дозволяє вибирати файли будь-якого типу («*All Files*|.»). Після цього користувачу пропонується вибрати файл у відповідному діалоговому вікні. У випадку, коли користувач вибирає файл та підтверджує свій вибір натисканням на кнопку «*OK*», відбуваються наступні дії:

Шлях до обраного файлу визначається та виводиться у текстове поле «*FilePathTBox*», дозволяючи користувачеві бачити вибраний ним файл. Далі

визначається назва файлу разом з його розширенням, яка потім зберігається у змінній «*_FileName*».

За процес завантаження файлів до *Google Drive* відповідає метод «*SaveBtn_Click*». Цей метод також здійснює перевірку коректності введених даних, інформує користувачів про результати завантаження та відповідно оновлює інтерфейс, що демонструється на рис. 4.16.

```
private void SaveBtn_Click(object sender, EventArgs e) {
  if (IsDataEnteringCorrect()) {
    string folderId = GetFilderId();
    // Отримуємо ідентифікатор папки з вибраного елемента комбо-бокса
    string fileName = UploadFileToDrive(FilePathTBox.Text, folderId);
    if (fileName != null) {
      _LogsProvider.InsertLogs(LoginForm.CurrentUser.UsersId,
        $"Були проведено завантаження. Файл: {fileName}", DateTime.Now);
      MessageBox.Show("Файл було успішно збережено на Google Drive!");
      SendFilePanel.Visible = true;
      UsersGridView.Visible = true;
    } else {
      MessageBox.Show("Файл не вдалося завантажити!");
    }
  }
}
```

Рис. 4.16. Реалізація методу «*SaveBtn_Click*»

Метод *SaveBtn_Click*, який активізується при натисканні на кнопку збереження, виконує ряд важливих дій:

– перевірка введених даних. Спочатку метод перевіряє правильність введення даних через функцію *IsDataEnteringCorrect*(). Ця функція може включати перевірку на наявність дійсного шляху до файлу та інші критерії валідності даних;

– визначення місця зберігання файлу. Далі метод отримує ідентифікатор папки з вибраного елемента в комбо-боксі, щоб визначити, де саме файл буде зберігатися на *Google Drive*;

– завантаження файлу на *Google Drive*. Використовуючи метод *UploadFileToDrive*, що приймає шлях до файлу та ідентифікатор папки, файл завантажується на *Google Drive*. У разі успішного завантаження (коли назва файлу не дорівнює *null*), в систему журналів додається запис про це завантаження через *_LogsProvider*, з відображенням ідентифікатора користувача, назви файлу та часу завантаження. Користувач отримує повідомлення про успішне збереження файлу на *Google Drive*, а панель відправки файлу та таблиця користувачів стають видимими.

У випадку невдачі (коли назва файлу дорівнює *null*), користувача інформують про невдале завантаження файлу.

Окрім того, було важливо забезпечити функціонал повідомлення інших користувачів системи про завантаження нового файлу. Для цього було реалізовано метод *SendMessage*, який відправляє електронні листи через *SMTP*-сервер *Google*, забезпечуючи користувачів системи актуальною інформацією про оновлення у хмарному сховищі.

У фрагменті коду, зображеному на рис. 4.17, демонструється механізм створення та відправки електронного листа.

```
private void SendMessage() {
  _SelectedNetworkCred = _NetworkCredProvider.LoadCredentials();
  // Створення нового екземпляру повідомлення
  var mailMessage = new MailMessage {
    From = new MailAddress(_SelectedNetworkCred.CredentialsEmail),
    Subject = "Зміна версії документу",
    Body = "Доброго дня, прохання ознайомитись із файлом "
    + _FileName +
    " який був завантажений користувачем " + LoginForm.CurrentUser.FIO,
    IsBodyHtml = true, // Встановити true, якщо тіло листа у форматі HTML
  };
  for (int i = 0; i < UsersGridView.RowCount; i++) {
    if (Convert.ToBoolean(UsersGridView["chk", i].Value)) {
      // Додавання адресатів
      mailMessage.To.Add(UsersGridView["Email", i].Value.ToString());
    }
  }
```

Рис. 4.17. Метод створення та відправлення електронного листа

Процес реалізації цього коду включає кілька ключових кроків:

– завантаження даних для авторизації у електронній пошті. Початковий крок полягає у завантаженні облікових даних за допомогою провайдера (*_NetworkCredProvider*). Це включає ім'я користувача, електронну адресу та інші важливі деталі, необхідні для авторизації та відправлення електронних листів;

– створення нового електронного листа. Наступним етапом є створення об'єкта «*MailMessage*», в якому визначаються основні атрибути листа, включно з відправником (використовуючи електронну адресу з завантажених даних), темою листа та текстом повідомлення. Текст листа містить інформацію про завантажений файл та ім'я користувача, який це зробив, а також форматується у вигляді *HTML*;

– вибір одержувачів листа. Код проходить по всім записам у таблиці користувачів, вибираючи електронні адреси тих, хто позначений прапорцем (*chk*). Для кожного такого користувача його електронна адреса додається до списку отримувачів. Це реалізується шляхом перевірки значення у кожному рядку через «*Convert.ToBoolean*» та вилучення електронної адреси з відповідного поля (*Email*)..

Цей код ефективно автоматизує процес інформування користувачів про важливі події чи зміни в системі, зокрема про оновлення документів, шляхом масової розсилки електронних листів.

На рис. 4.18 представлено код для налаштування та створення клієнта *SMTP*, який забезпечує надійну та безпечну відправку електронних листів через сервери *Google* з використанням стандартних протоколів безпеки.

```
// Створення клієнта SMTP з відомостями про сервер
var smtpClient = new SmtpClient("smtp.gmail.com") // SMTP сервер
{
  Port = _SelectedNetworkCred.Port, // Стандартний порт для SMTP
  // облікові дані для авторизації на SMTP сервері
  Credentials = new NetworkCredential(_SelectedNetworkCred.CredentialsEmail,
  _EncryptData.Decrypt(_SelectedNetworkCred.CredentialsPass)),
  // Включення SSL для шифрування з'єднання
  EnableSsl = _SelectedNetworkCred.EnableSsl,
};
```

Рис. 4.18. Налаштування клієнта *SMTP*

Процес включає ініціацію нового об'єкта «*SmtpClient*», який відіграє ключову роль у відправці електронних листів. Код використовує *SMTP*-сервер «smtp.gmail.com» від *Google* для забезпечення цього процесу. Вказується порт для з'єднання з *SMTP*-сервером, який відповідає стандартному порту, використовуваному для *SMTP*-з'єднань. Значення цього порту отримується з об'єкта «*_SelectedNetworkCred*».

Для авторизації на *SMTP*-сервері використовуються облікові дані, які включають електронну адресу та дешифрований пароль. Пароль дешифрується за допомогою методу «*Decrypt*» з класу «*_EncryptData*», що гарантує безпечне використання парольної інформації.

Додатково, для підвищення безпеки передачі даних на *SMTP*-сервер активується *SSL* (*Secure Sockets Layer*). Це дозволяє шифрувати дані, що передаються на сервер, значно підвищуючи рівень безпеки в процесі комунікації.

Для надання користувачам системи «Репозиторій» можливості додавати файли як вкладення до електронних листів та відправляти їх, було реалізовано спеціальний код, який представлено на рис. 4.19.

```
// Перевірка, чи шлях до файлу не є порожнім
if (!string.IsNullOrEmpty(FilePathTBox.Text)) {
  // Створення об'єкта Attachment
  var attachment = new Attachment(FilePathTBox.Text);
  // Додавання файлу до листа
  mailMessage.Attachments.Add(attachment);
}
```

Рис. 4.19. Механізм прикріплення файлів до електронних листів

У цьому коді перш за все проводиться перевірка наявності шляху до файлу у текстовому полі «*FilePathTBox*». Це дозволяє запобігти додаванню віртуальних файлів у вкладення до листа. У випадку, коли шлях до файлу існує та вказаний правильно, створюється об'єкт «*Attachment*», який представляє собою файл для прикріплення. Цей файл, зазначений у полі «*FilePathTBox*», стає частиною об'єкта «*Attachment*». Потім цей об'єкт включається до колекції вкладень електронного листа (*mailMessage*), перетворюючи файл на вкладення листа.

Додатково було реалізовано код для відправлення електронного листа, що включає обробку можливих помилок та звільнення використаних ресурсів після завершення процесу. Цей аспект коду відображений на рис. 4.20.

```
// Відправлення листа
smtpClient.Send(mailMessage);
MessageBox.Show("Лист успішно відправлено.");
UsersGridView.Visible = false;
InformBtn.Visible = false;
SendFilePanel.Visible = false;
FilePathTBox.Text = string.Empty;
```

Рис. 4.20. Процес відправлення електронного листа

Таким чином, завдяки цим реалізаціям, користувачі системи «Репозиторій» мають можливість не лише завантажувати файли до хмарного сховища, але й ефективно сповіщати інших користувачів системи про зміни або оновлення, що значно спрощує процеси управління та обміну документами.

4.4. Імплементація функціональних модулів системи

Завершивши успішну інтеграцію бази даних у розроблювану систему «Репозиторій» на платформі *Visual Studio 2022*, основний фокус був зосереджений на розробці бізнес-логіки. Це включало розробку спеціалізованих алгоритмів для обробки метаданих файлів, що є важливою складовою в системі управління документами.

Початковий крок у цьому процесі полягав у модифікації змінної *CONNECT* у файлі конфігурації *App.config*. Ця змінна містить ключові параметри для з'єднання з базою даних, що є фундаментальною частиною архітектури системи. Візуалізацію цієї частини можна переглянути на рис. 4.21.

```
<!-- Підключення до бази даних -->
<appSettings>
  <add key="CONNECT" value="Data Source=(LocalDB)\MSSQLLocalDB;
       AttachDbFilename=|DataDirectory|\DB.mdf;Integrated Security=True" />
</appSettings>
```

Рис. 4.21. Конфігурація з’єднання з «*MSSQLLocalDB*»

У рамках розробки системи «Репозиторій» було застосовано простір імен *System.Data.SqlClient* з стандартної бібліотеки *.NET* для забезпечення ефективної роботи з реляційною базою даних. *System.Data.SqlClient* спеціалізується на взаємодії з *SQL Server* і включає в себе комплекс класів, які охоплюють весь спектр взаємодії з базою даних – від налагодження з'єднань до виконання *SQL*-запитів. Це забезпечує глибокий та безпечний контроль над даними, мінімізуючи ризики помилок під час їх обробки.

Інтеграція компоненту *MenuStrip* у користувацький інтерфейс системи є ключовим аспектом, що покращує зручність та ефективність використання. *MenuStrip*, який є стандартним елементом управління в *Windows Forms*, дозволяє створювати адаптивні меню з різноманітними опціями, що полегшує навігацію для користувачів та надає швидкий доступ до ключових функцій системи. Це демонструється на рис. 4.22.

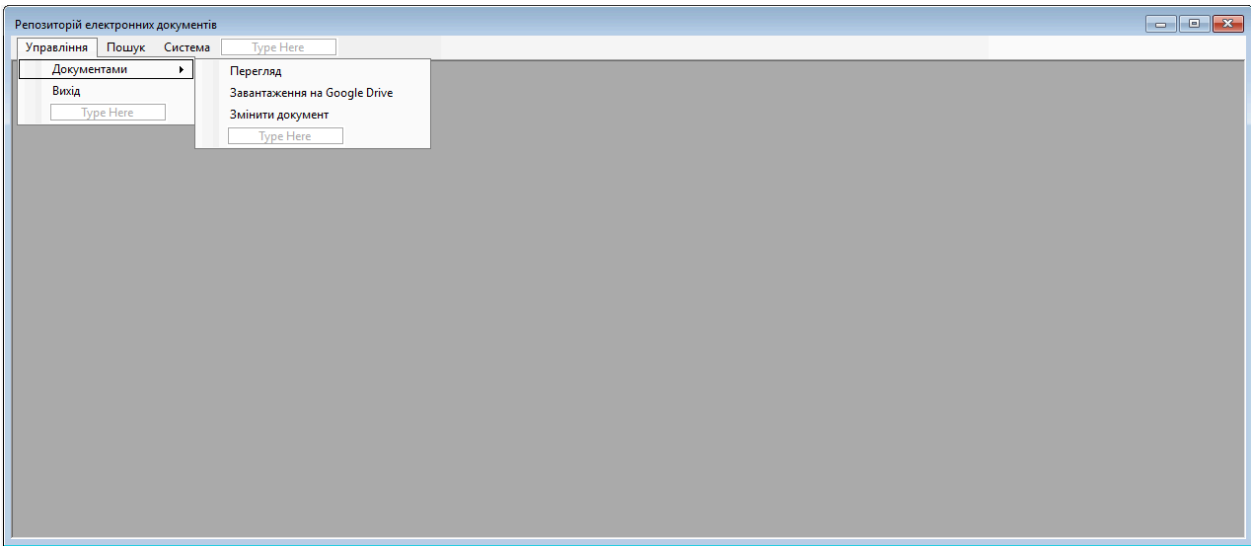

Рис. 4.22. Додавання головного меню в інтерфейс

У системі «Репозиторій» реалізовано механізм управління відкритими формами через пункти головного меню. Кожен пункт меню пов'язаний з активацією відповідної форми, дозволяючи автоматизувати процес закриття поточного вікна перед відкриттям нового, що запобігає графічним конфліктам та перешкодам між вікнами.

Ця уніфікована система управління формами застосовується до всіх пунктів меню, забезпечуючи консистентність взаємодії користувача з різними частинами програми. Така однорідність сприяє зрозумілості і легкості використання системи. Докладний опис цього механізму та його реалізацію можна знайти в відповідному розділі коду, що відображено на рис. 4.23.

```
private void налаштуванняToolStripMenuItem_Click(object sender, EventArgs e) {
    if (LoginForm.CurrentUser.RoleId == 1) {
        CloseAllWindows();
        OptionsForm optionsForm = new OptionsForm();
        optionsForm.MdiParent = this;
        optionsForm.WindowState = FormWindowState.Maximized;
        optionsForm.Show();
    } else {
        MessageBox.Show(NamesMy.MessageBoxExaption.YouDontHavePermission);
    }
}
```

Рис. 4.23. Код створення екземплярів форм

Архітектурний підхід, використаний у системі «Репозиторій», спрощує процес розробки завдяки можливості повторного використання коду для управління вікнами та меню, одночасно забезпечуючи стабільність та надійність системи протягом її

використання. Завершення основних етапів розробки дало змогу створити класи для взаємодії з реляційною базою даних, серед яких метод «*InsertBatchDriveFileList*», представлений на рис. 4.24, виконує важливу функцію масового додавання даних про файли з хмарного сховища до локальної бази даних.

```
public void InsertBatchDriveFileList(List<DriveFile> DriveFileList) {
  DeleteAllFromDriveFiles();
  SqlConnection connection = new SqlConnection(_ConnString);
  string query = "INSERT into DriveFiles (Id, Name, MimeType, Level) " +
      "VALUES (@Id, @Name, @MimeType, @Level)";
  SqlCommand command = new SqlCommand(query, connection);
  connection.Open();
  for (int i = 0; i < DriveFileList.Count; i++) {
    command.Parameters.AddWithValue("@Id", DriveFileList[i].Id);
    command.Parameters.AddWithValue("@Name", DriveFileList[i].Name);
    command.Parameters.AddWithValue("@MimeType", DriveFileList[i].MimeType);
    command.Parameters.AddWithValue("@Level", DriveFileList[i].Level);
    command.ExecuteNonQuery();
    command.Parameters.Clear();
  }
  connection.Close();
}
```

Рис. 4.24. Код методу «*InsertBatchsDriveFileList*»

Метод «*InsertBatchsDriveFileList*» забезпечує масове додавання інформації про файли з хмарного сховища до бази даних. Це включає створення *SQL*-запиту з параметрами для зниження ризиків *SQL*-ін'єкцій та забезпечення безпеки системи. Перед вставкою даних відбувається очищення відповідної таблиці, щоб уникнути дублювання інформації. Для виконання *SQL*-запитів створюється нове підключення до бази даних, а в межах цього підключення – об'єкт *SqlCommand*. Процес додавання даних реалізується через цикл, в якому параметри *SQL*-запиту заповнюються відповідно до даних кожного файлу зі списку *DriveFileList*, після чого виконується запит.

Цей метод триває до тих пір, поки всі елементи списку не будуть внесені до бази. Після завершення цього процесу підключення до бази даних закривається. Такий підхід забезпечує ефективну та безпечну взаємодію з базою даних, гарантуючи високий рівень захисту даних та надійності їх обробки.

Метод «*GetAllDriveFiles*» є ключовим елементом в системі «Репозиторій», забезпечуючи вилучення актуального списку файлів із локальної бази даних, яка синхронізована з хмарним сховищем. Візуалізація цього методу представлена на рис. 4.25.

```
public List<DriveFile> GetAllDriveFiles() {
  List<DriveFile> DriveFileList = new List<DriveFile>();
  string sqlExpression = "SELECT * FROM DriveFiles";
  using (SqlConnection connection = new SqlConnection(_ConnString)) {
    connection.Open();
    SqlCommand command = new SqlCommand(sqlExpression, connection);
    SqlDataReader reader = command.ExecuteReader();

    if (reader.HasRows) {
      while (reader.Read()) {
        DriveFile selectedFile = new DriveFile();
        selectedFile.DriveFileId = Convert.ToInt32(reader["DriveFileId"]);
        selectedFile.Id = reader["Id"].ToString();
        selectedFile.Name = reader["Name"].ToString();
        selectedFile.MimeType = reader["MimeType"].ToString();
        selectedFile.Level = Convert.ToInt32(reader["Level"]);
        DriveFileList.Add(selectedFile);
      }
    }
    reader.Close();
  }
  return DriveFileList;
}
```

Рис. 4.25. Реалізація методу «*GetAllDriveFiles*»

Цей метод ініціює процес вибірки даних, запускаючи *SQL*-запит до таблиці «*DriveFiles*» у локальній базі даних. Застосування конструкції «*using*» для об'єкта *SqlConnection* гарантує автоматичне закриття з'єднання після завершення роботи, підвищуючи таким чином надійність системи.

Після активації підключення до бази даних, створюється об'єкт *SqlCommand*, який виконує запит. За допомогою *SqlDataReader*, метод послідовно переглядає всі записи, отримані в результаті запиту. Для кожного з них створюється новий екземпляр класу *DriveFile*, який заповнюється відповідними даними. Отримані екземпляри *DriveFile* додаються до колекції *DriveFileList*, яка в кінці методу повертається як результат.

Цей підхід дозволяє системі ефективно та безпечно обробляти великі обсяги даних, забезпечуючи вибірку всієї потрібної інформації з бази даних. Метод включає заходи безпеки, такі як параметризовані запити та контроль закриття з'єднання з базою, що сприяє забезпеченню консистентності та цілісності даних у системі управління електронними документами.

Після реалізації ключових класів в системі «Репозиторій», наступним етапом розробки стало створення користувацьких інтерфейсів для забезпечення взаємодії користувачів із системою. Особлива увага була зосереджена на розробці інтерфейсу

модуля пошуку файлів у базі даних, який відіграє важливу роль у користувацькому інтерфейсі системи. Цей модуль, показаний на рис. 4.26, спрощує процедуру пошуку документів і забезпечує користувачам легкий доступ до потрібної інформації.

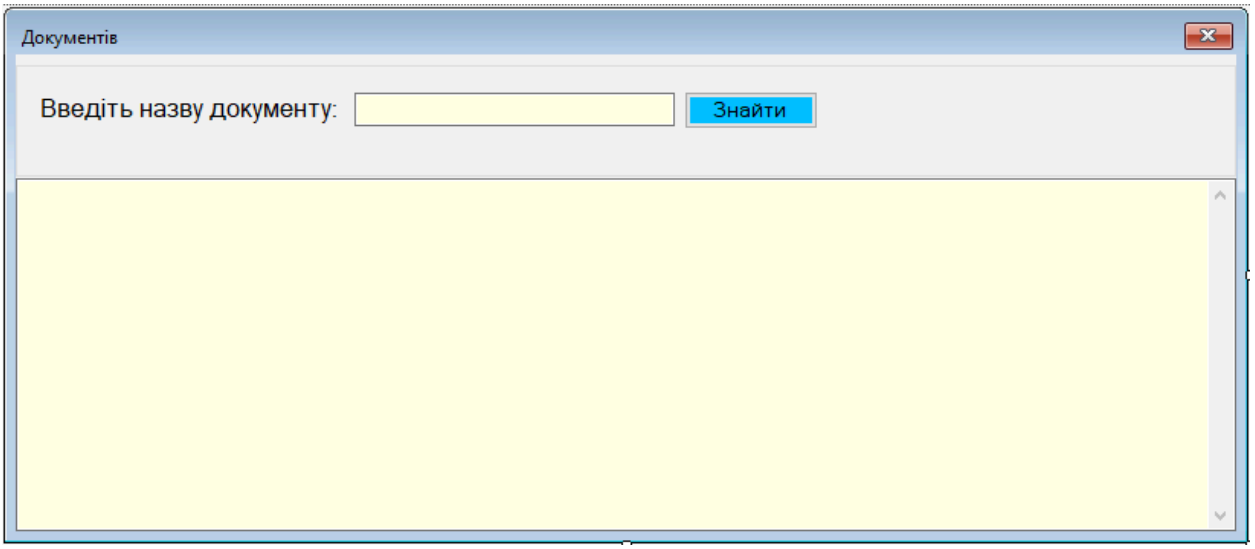

Рис. 4.26. Інтерфейс форми для пошуку документів

Форма «*SearchFilesForm*» має ключове значення для реалізації функції пошуку файлів у системі управління електронними документами. Ця форма служить як графічний інтерфейс, що надає користувачам простий і інтуїтивно зрозумілий доступ до бази даних документів, дозволяючи їм швидко знаходити необхідні файли. Детальний опис роботи та функціональності цієї форми надається у відповідному розділі.

Процес ініціалізації форми включає виклик методу «*InitializeComponent*», який відповідає за конфігурацію та візуалізацію різних компонентів інтерфейсу. Цей метод є стандартним у середовищі розробки і генерується автоматично, сприяючи зручності та ефективності розробки інтерфейсу.

У формі «*SearchFilesForm*» використовується клас «*SearchBLL*» для надання бізнес-логіки пошуку файлів. Екземпляр цього класу, позначений як «*_RaportBLL*», ініціюється під час ініціалізації форми і використовується для виклику методів пошуку, що дозволяє користувачам взаємодіяти з базою даних. Це демонструється на рис. 4.27.

```
SearchBLL _RaportBLL = new SearchBLL();
List<DriveFile> _DriveFileList = new List<DriveFile>();
```

Рис. 4.27. Ініціація класу «*SearchBLL*»

Коли користувач використовує кнопку пошуку, запускається метод «*SearchBtn_Click*», зображений на рис. 4.28. У цьому методі функція *GetRaportForClientFIO* з класу *_RaportBLL* використовує текст з поля пошуку *SearchTBox* для формування списку файлів, які відповідають заданим критеріям.

```
private void SearchBtn_Click(object sender, EventArgs e) {
  _DriveFileList = _RaportBLL.GetRaportForClientFIO(SearchTBox.Text);
  GetRaport(_DriveFileList);
}
```

Рис. 4.28. Функціонал методу «*SearchBtn_Click*»

Далі для відображення результатів пошуку у форматованому вигляді використовується метод «*GetRaport*» (рис. 4.29). Він виконує задачу заповнення текстового поля «*RaportTBox*», розміщуючи інформацію про знайдені файли у вигляді таблиці. Для цього застосовуються різні методи форматування тексту, щоб забезпечити зручне та зрозуміле представлення даних.

```
public void GetRaport(List<DriveFile> DriveFileList) {
  // Очищення та ініціалізація текстового поля
  RaportTBox.Text = "Пошук по документах:\r\n";
  // Заголовок таблиці
  RaportTBox.Text += String.Format("{0,5}|{1,-40}|{2,-20}\r\n", "№", "Ім'я файлу", "Тип файлу");

  // Ітерація по списку DriveFile
  for (int i = 0; i < DriveFileList.Count; i++) {
    // Формування рядка для кожного елемента списку
    string raportString = String.Format("{0,5}|{1,-40}|{2,-20}\r\n",
      DriveFileList[i].Number,
      DriveFileList[i].Name,
      DriveFileList[i].MimeType
    );
    // Додавання сформованого рядка до текстового поля
    RaportTBox.Text += raportString;
  }
 }
```

Рис. 4.29. Код методу «*GetRaport*»

Метод в формі «*SearchFilesForm*» для пошуку файлів у репозиторії включає наступні ключові етапи:

– ініціалізація та очищення текстового поля. Спочатку метод очищає текстове поле «*RaportTBox*» і встановлює початковий текст, що інформує про процес пошуку документів;

– формування заголовка таблиці. Далі в текст додається форматований заголовок таблиці, який включає в себе номер, назву та тип файлу. Це здійснюється за допомогою методу форматування рядків, щоб забезпечити чіткість і зрозумілість даних;

– ітерація по списку файлів. Метод перебирає кожен елемент у списку «*DriveFileList*». Для кожного файлу у списку формується рядок з інформацією, що включає номер файла, його назву та тип. Форматування кожного рядка гарантує легкість сприйняття інформації;

– додавання інформації до текстового поля. Всі сформовані рядки послідовно додаються до текстового поля «*RaportTBox*», що забезпечує систематичне відображення інформації про кожен файл зі списку.

Форма «*SearchFilesForm*» таким чином забезпечує комплексну та ефективну функціональність пошуку файлів у системі репозиторію, спрощуючи взаємодію користувача з програмним забезпеченням. Її архітектура орієнтована на забезпечення гнучкості, модульності та легкості в подальшому розвитку та обслуговуванні програми.

4.5. Функціональне та модульне тестування системи

Тестування програмного забезпечення є критично важливим етапом у процесі розробки, оскільки це дозволяє переконатися, що програма задовольняє всі поставлені вимоги і виконує свої функції ефективно. Цей процес включає ретельну перевірку програми, що допомагає виявити та виправити помилки, тим самим поліпшуючи якість та ефективність продукту.

Під час тестування звертається увага на такі ключові аспекти, як продуктивність, функціональність, стабільність та безпека програми. Це необхідно для виявлення потенційних проблем та їх вирішення, а також для переконання в тому, що програма відповідає усім встановленим вимогам і очікуванням користувачів.

У контексті розробки інформаційної системи, функціональне тестування було проведено після завершення основної фази розробки. Цей етап охоплював детальну перевірку основних функцій та можливостей програми, з метою забезпечення її стабільності та надійності. Для цього були розроблені спеціалізовані тест-кейси, які включали конкретні сценарії, розроблені на основі функціональних вимог. Ці тест-кейси допомогли ідентифікувати та виправити можливі помилки та недоліки, що в кінцевому підсумку сприяло підвищенню якості та ефективності системи перед її впровадженням у використання.

Сценарій тесту №1. Валідація авторизації користувача в системі:

– ініціювати процес авторизації, відкривши відповідну форму;

– ввести валідні дані для логіна та пароля користувача;

– переконатись, що користувачу надано доступ до основної форми додатка.

Очікуваний результат: Користувач успішно авторизується і переходить до основної форми застосунку (рис. 3.30).

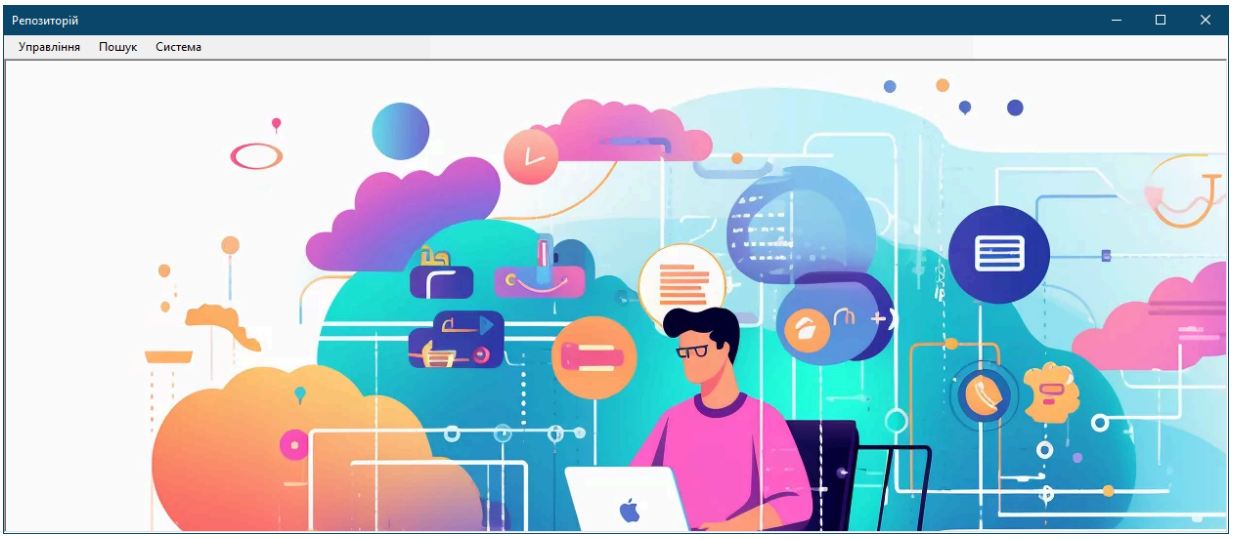

Рис. 4.30. Сценарій «Валідація авторизації користувача в системі»

Сценарій тесту №2. Синхронізація електронних документів репозиторію із базою даних.

Ціль: Перевірити, що синхронізація електронних документів з *Google Drive* у базу даних репозиторію працює коректно і зберігає дані у відповідності з отриманими від хмарного сховища.

Кроки сценарію:

– відкрити форму «Налаштування репозиторію»;

– натиснути кнопку «Синхронізація» для запуску процесу синхронізації;

– авторизація у *Google Drive*. Переконатися, що авторизація на *Google Drive* відбувається без помилок;

– збір даних з *Google Drive*. Перевірити, чи запит до *Google Drive API* виконується без помилок;

– перевірити, чи отримано відповідь із списком файлів;

– перевірити, що кожен файл та папка коректно відображаються у *TreeView*;

– перевірити, що всі файли з хмарного сховища були коректно збережені у локальну базу даних;

– переконатися, що в журналі подій системи з'являється запис про успішну синхронізацію.

Очікуваний результат: Всі етапи проходять без помилок. Документи з *Google Drive* відображаються у деревоподібній структурі на інтерфейсі, інформація про файли з хмарного сховища коректно зберігається у локальній базі даних, а в журналі подій з'являється запис про успішну синхронізацію (рис. 3.31).

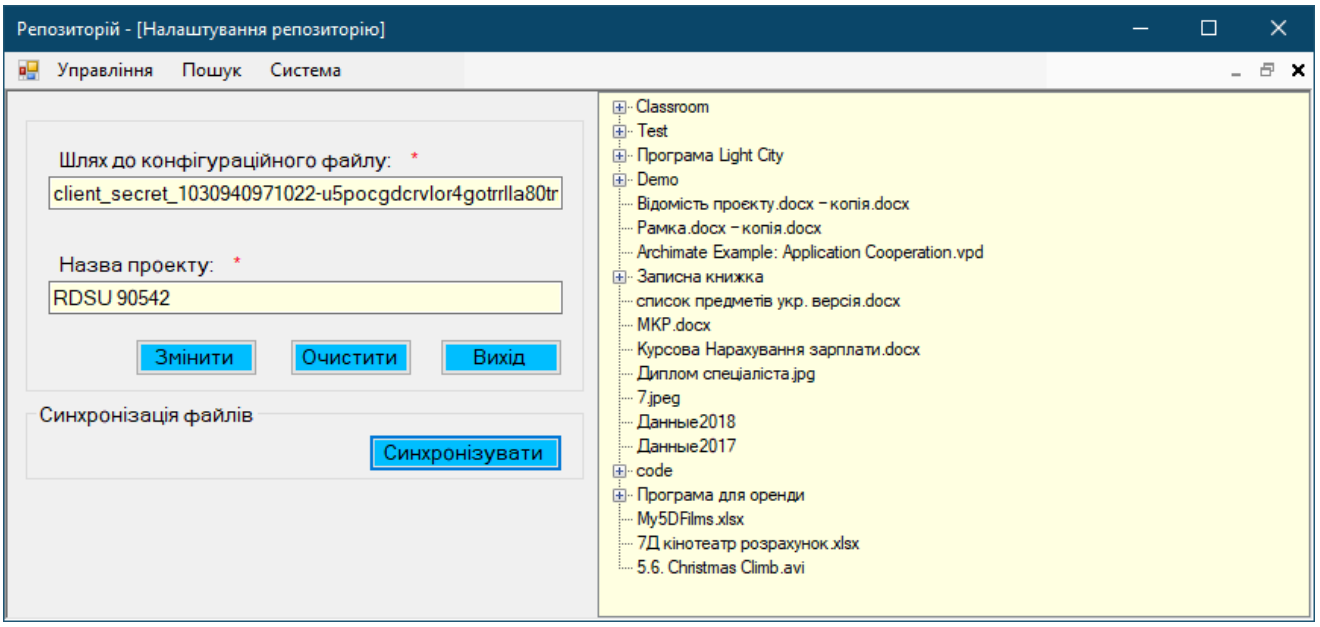

Рис. 4.31. Сценарій «Синхронізація електронних документів репозиторію»

Сценарій тесту №3: Завантаження файлів на *Google Drive* та інформування користувачів.

Ціль: Перевірити функціональність форми для завантаження файлів на *Google Drive* та відправлення повідомлень користувачам про завантаження файлів.

Кроки сценарію:

– відкриття форми для заміни документу у репозиторії;

– введення назви файлу у полі пошуку файлів та натискання кнопки «Знайти»;

– вибір файлу для завантаження. Натиснути на поле вибору файлу, щоб відкрити вікно вибору файлу;

– завантаження файлу на *Google Drive*. Натиснути кнопку «Завантажити». Перевірити, чи файл було успішно завантажено на *Google Drive* у вибрану папку та переконатися, що з'являється повідомлення про успішне завантаження.

– інформування користувачів. Вибрати одного або декілька користувачів зі списку;

– натиснути кнопку інформування для відправки електронного листа через *SMTP*-сервер. Переконатися, що листи були успішно відправлені, і користувач отримує повідомлення про успішну або невдалу відправку.

Очікуваний результат: Файл правильно завантажується на *Google Drive* у вказану папку. Користувач отримує повідомлення про успіх завантаження. Листи успішно відправляються обраним користувачам, і користувач отримує відповідне повідомлення про статус відправки (рис. 3.32).

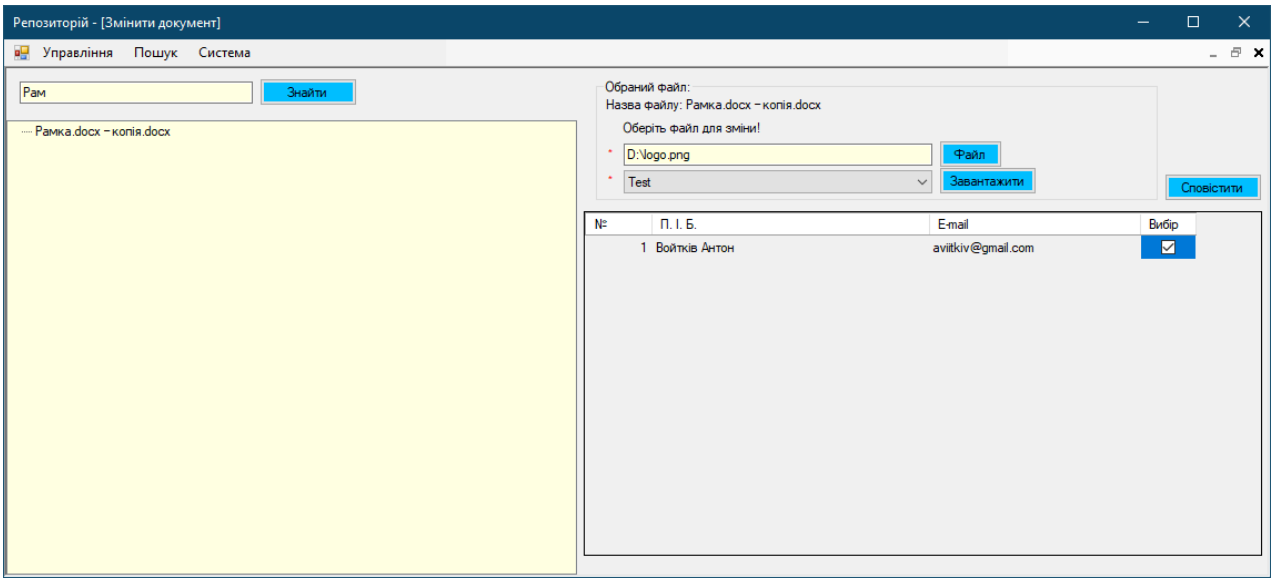

Рис. 4.32. Сценарій «Заміна документу у репозиторії»

На фінальній стадії розробки програмного продукту «Репозиторій», особлива увага була зосереджена на ретельному аналізі якості коду та його відповідності заданим функціональним вимогам. Розробники використали інструмент юніт-тестування у *Visual Studio*, створивши спеціалізований проект для цього, відомий як «*Unit Test Project*». Ця стратегія дозволила детально перевірити кожен компонент системи у контрольованому середовищі, підвищуючи точність тестування.

Головною ціллю було тестування логіки методів, їх взаємодії в системі та реакції на різні вхідні дані. Розроблено набір юніт-тестів, що дозволив детально оцінити функціональність кожного аспекту системи. Вивчення результатів цих тестів забезпечило чітке візуальне представлення функціональної ефективності системи.

Цей метод тестування виявив потенційні проблеми та неузгодженості у роботі окремих модулів, надавши можливість розробникам своєчасно оптимізувати код. Завдяки цьому, процес розробки став ефективнішим, забезпечивши вищий рівень надійності та якості кінцевого продукту. В результаті, програмний комплекс «Репозиторій» був оптимізований для стабільної роботи в реальних умовах використання.

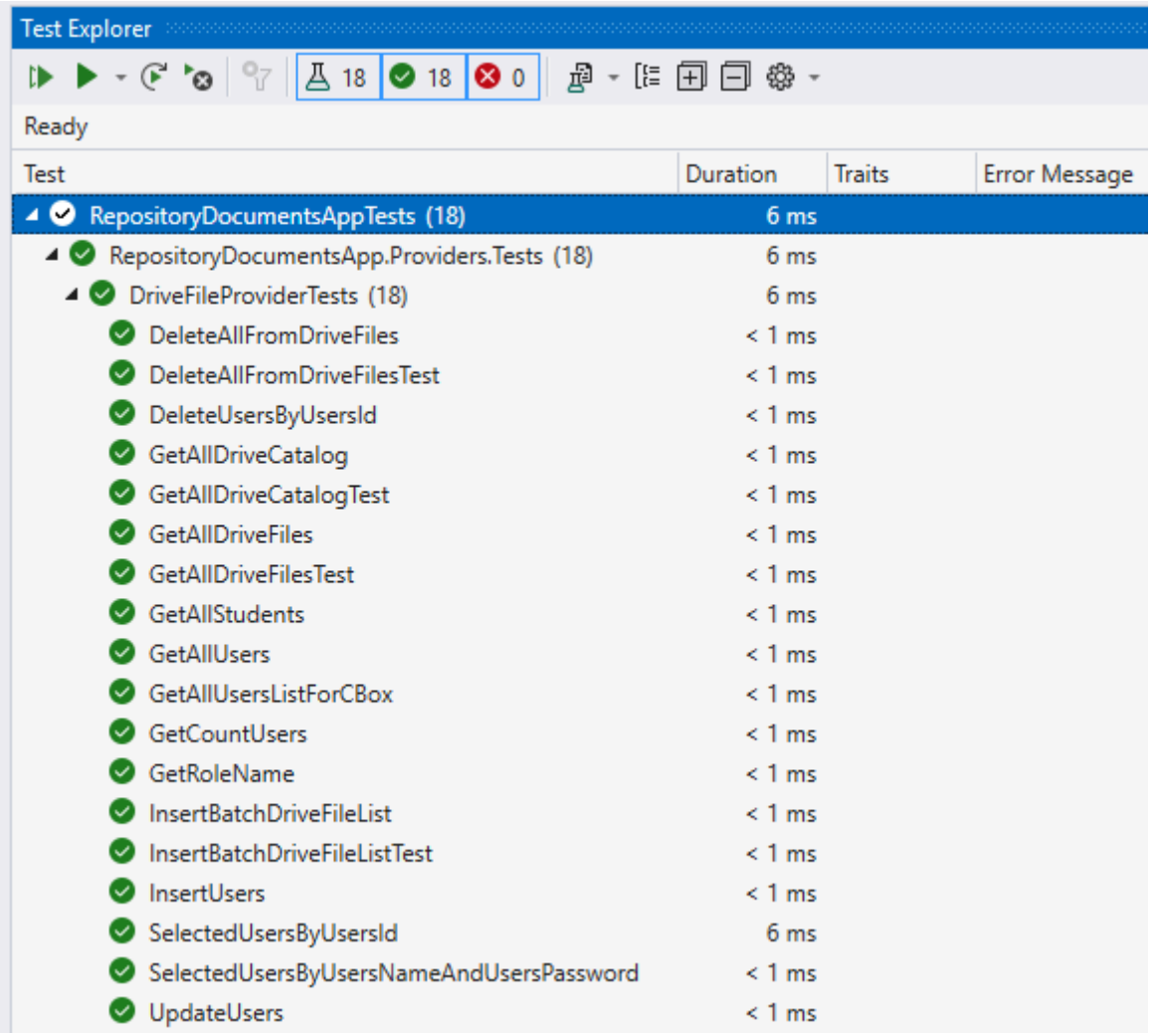

Рис. 4.33. Результати модульного тестування

4.6. Висновки до розділу

У цьому розділі представлено розробку системи для зберігання та управління електронними документами. Проект акцентує на ключових компонентах: архітектурі, структурі бази даних, інтеграції з хмарними сервісами, реалізації важливих функцій, а також тестуванні та валідації системи. Архітектура, побудована на трьох рівнях, забезпечує міцну основу для гнучкості, масштабованості, безпеки та ефективного управління системою. База даних сконцентрована навколо чотирьох основних таблиць, які разом формують потужний механізм для управління документами та метаданими.

Система інтегрована з *Google Drive* через *NuGet*-пакет *Google.Apis.Drive.v3*, що значно розширює її функціональність, додаючи можливості зберігання та

доступу до документів у хмарі. Функціональні модулі, які включають меню навігації, управління вікнами та інтеграцію з *SQL Server*, є ключовими елементами, що забезпечують зручність та ефективність використання системи.

Детальне тестування відіграє важливу роль у забезпеченні надійності та якості системи, дозволяючи виявити та виправити будь-які можливі проблеми до її запуску. Ці елементи разом формують міцну основу для майбутнього розвитку та вдосконалення системи, гарантуючи її стабільність і ефективність у роботі з електронними документами.

ВИСНОВКИ

В ході виконання магістерської роботи була створена об'ємна система для управління електронними документами. Ця система, орієнтована на зберігання, організацію та забезпечення доступу до різноманітних файлів та документів, втілює глибокий комплексний підхід. Вона структурована на основі трьохрівневої архітектури, яка відкриває широкі можливості для адаптації, розширення та ефективного управління системою. Ця архітектурна модель сприяє не лише гнучкості системи, але й спрощує її технічне обслуговування та масштабування, роблячи її ідеальною для комплексного управління електронними документами.

Перший розділ магістерської роботи присвячено детальному аналізу архітектури хмарних обчислень, їх різноманітних моделей розгортання, а також методик управління даними в рамках хмарних середовищ. Значна увага зосереджена на вивченні впливу хмарних технологій на сталість та екологічність, а також на розгляді потенціалу інтеграції штучного інтелекту в хмарні обчислення. Подано детальний огляд основних моделей надання послуг у сфері хмарних обчислень: програмне забезпечення як послуга (*SaaS*), платформа як послуга (*PaaS*), та інфраструктура як послуга (*IaaS*). Обговорюються головні схеми розгортання хмарних сервісів, включаючи публічні, гібридні та приватні хмари.

В рамках розділу виокремлені ключові аспекти управління даними в хмарних середовищах, що включають зберігання даних, керування їх життєвим циклом, заходи безпеки та шифрування, а також стратегії резервного копіювання та відновлення. Окрема увага приділена викликам, пов'язаним з безпекою та конфіденційністю даних при використанні хмарних сервісів, з акцентом на потенційні ризики та методи їх мінімізації.

У другому розділі здійснено ґрунтовний аналіз методів резервного копіювання об'ємних даних, з акцентом на порівняльний огляд традиційних і сучасних підходів, їх переваг і недоліків, особливо в контексті використання у хмарних сервісах. В деталях вивчені різні методи резервування - повне, диференційне, та

інкрементальне, включаючи їх специфічні особливості та практичне використання. Особлива увага приділена аспектам хмарного резервного копіювання, як то масштабованість сховищ, гнучкість ресурсів, автоматизації процесів, швидкість передачі даних, безпеки, та ефективності дедуплікації даних.

Проведений аналіз ефективності рішень для резервного копіювання в хмарних середовищах включав розгляд таких платформ, як *Google Drive, Microsoft Azure*, та *Amazon Web Services*. Зокрема, було вивчено технічні особливості та методи інтеграції систем резервного копіювання із зазначеними хмарними сервісами. В результаті аналізу виявлено, що *Google Drive* виявився найбільш відповідним для потреб резервного копіювання завдяки його інтуїтивній навігації, тісній інтеграції з іншими продуктами *Google*, а також завдяки ефективним механізмам синхронізації та спільного використання даних. Це забезпечує високу гнучкість та оперативність процесів резервного копіювання, роблячи *Google Drive* ідеальним вибором для сучасного резервування даних.

Третій розділ дослідження був присвячений ґрунтовному аналізу та проектуванню системи електронного документообігу, обіймаючи ключові аспекти її створення. Вихідним пунктом стало чітке визначення вимог до системи, охоплюючи функціональні та нефункціональні атрибути, що забезпечило глибоке розуміння очікувань від системи та стандартів її продуктивності. Детально були визначені основні дійові особи та їх цілі в системі, що дозволило точно відповісти на запити користувачів. У процесі розробки були створені діаграми прецедентів, які відіграли вирішальну роль у візуалізації взаємодій між акторами та компонентами системи, надаючи детальне розуміння їх взаємозв'язків. Також були розроблені діаграми бізнес-процесів, які включали в себе діяльнісні схеми основних операцій системи, відображаючи авторизацію користувачів, завантаження та пошук файлів. Діаграми послідовностей були використані для демонстрації процесів налаштування репозиторію та синхронізації даних, що дало додаткове уявлення про взаємодії в системі.

Четвертий розділ роботи був присвячений детальній розробці системи для управління електронними документами, з особливим фокусом на її ключових

аспектах. Основу системи складає її трьохрівнева архітектура, яка сприяє її гнучкості, масштабованості, безпеці, а також ефективності в управлінні. Центральним елементом системи є її база даних, структурована навколо чотирьох основних таблиць, що разом формують комплексний механізм для обробки документів та їх метаданих.

Інтеграція системи з хмарним сервісом *Google Drive* через *NuGet*-пакет *Google.Apis.Drive.v3* значно розширює її можливості, зокрема щодо зберігання та доступу до документів у хмарі. Система включає ряд функціональних модулів, серед яких меню навігації, управління вікнами та інтеграція *з SQL Server*, що забезпечують її ефективність та зручність використання.

Ключовим етапом у розробці системи було її детальне тестування, спрямоване на забезпечення високої надійності та якості. Тестування допомогло виявити та виправити потенційні недоліки системи перед реалізацією її останньої версії.

Розроблена система управління електронними документами становить значний потенціал для різноманітних сфер застосування, особливо там, де критично важливими є організація, безпека та швидкий доступ до інформації. Її гнучка архітектура та інтеграція з хмарними сервісами, як-от *Google Drive*, робить її ідеальною для великих та середніх підприємств, зокрема її можуть використовувати:

– корпоративний сектор. Великі та середні компанії можуть використовувати систему для ефективного управління корпоративними документами, контрактами, фінансовими звітами та іншою внутрішньою документацією;

– освітні установи. Університети, коледжі та школи можуть застосовувати систему для організації навчальних матеріалів, дослідницьких робіт, адміністративних документів та звітності;

– державні організації. Система може бути корисна для електронного документообігу в державних структурах, спрощуючи процеси обробки та зберігання офіційних документів;

– юридичні фірми та консалтингові компанії. Ці організації можуть застосовувати систему для зберігання та управління юридичними документами, договорами та консультативними матеріалами;

– науково-дослідницькі інститути. Для зберігання наукових публікацій, дослідницьких даних та інших академічних матеріалів;

– стартапи та IT-компанії. Інноваційні технологічні компанії можуть використовувати систему для управління проектною документацією, кодом та розробками.

Кожна з цих сфер вимагає індивідуального підходу до управління документами, і розроблена система здатна задовольнити ці потреби завдяки своїй адаптивності та високому рівню кастомізації.

СПИСОК БІБЛІОГРАФІЧНИХ ПОСИЛАНЬ ВИКОРИСТАНИХ ДЖЕРЕЛ

1. Бойченко С.В., Іванченко О.В. Положення про дипломні роботи (проекти) випускників Національного авіаційного університету. – К.: НАУ, 2017. – 63 с.

2. ДСТУ ГОСТ 3008-95 «Документація. Звіти у сфері науки і техніки. Структура і правила оформлення».

3. Матросова Н.М. Особливості архітектури систем на базі хмарних технологій: Інститут інформаційних технологій і засобів навчання НАПН України. 2013. 189с.

4. Що таке хмарні обчислення і як вони працюють?. *URL: https://experience.dropbox.com/uk-ua/resources/what-is-the-cloud* (дата звернення 20.12.2023).

5. *Hashemipour, S., Ali, M. (*2020*). Amazon Web Services (AWS) – An Overview of the On-Demand Cloud Computing Platform. In: Miraz, M.H., Excell, P.S., Ware, A., Soomro, S., Ali, M. (eds) Emerging Technologies in Computing. iCETiC* 2020*. Lecture Notes of the Institute for Computer Sciences, Social Informatics and Telecommunications Engineering, vol* 332*. Springer, Cham . URL: https://doi.org/10.1007/978-3-030-60036-5_3* (дата звернення 20.12.2023).

6. *What's ahead for cloud in 2023? Taking control of complexity, observability, and data . URL: https://www2.deloitte.com/us/en/pages/consulting/articles/2023-whats-new-whats-trendy-and-whats-next-for-cloud-Deloitte-on-cloud-podcast-cloud-complexity-observability-SRE-data-management-cloud-trends-2023.html*. (дата звернення 20.12.2023).

7. *Cloud computing trends* 2023*: Top predictions, stats, and growth drivers. URL: https://www.the-future-of-commerce.com/2022/11/09/cloud-computing-trends-2023/* (дата звернення 20.12.2023).

8. Current Development, Challenges and Future Trends in Cloud Computing: A Survey. (IJACSA) International Journal of Advanced Computer Science and Applications,Vol. 14*, No.* 3, 2023. 11 с.

9. Глинський Я.М. Штучний інтелект. Інтелектуальні роботи . Львів. 2002. 168 с.

10. Michael Cusumano. 2010. Cloud computing and SaaS as new computing platforms. Commun. URL: https://doi.org/10.1145/1721654.1721667(дата звернення 20.12.2023).

11. Qian, L., Luo, Z., Du, Y., Guo, L. (2009). Cloud Computing: An Overview. In: Jaatun, M.G., Zhao, G., Rong, C. (eds) Cloud Computing. CloudCom 2009. Lecture Notes in Computer Science, vol 5931. Springer, Berlin, Heidelberg. URL: https://doi.org/10.1007/978-3-642-10665-1_63 (дата звернення 20.12.2023).

12. Iosup, A., Prodan, R., Epema, D. (2014). IaaS Cloud Benchmarking: Approaches, Challenges, and Experience. In: Li, X., Qiu, J. (eds) Cloud Computing for Data-Intensive Applications. Springer, New York, NY. URL: https://doi.org/10.1007/978-1-4939-1905-5_4 (дата звернення 20.12.2023).

13. Public vs Private vs Hybrid vs Community - Cloud Computing: A Critical Review. I.J. Computer Network and Information Security, 2014, 3, 29с.

14. *Aishwarya Srinivasan, Md Abdul Quadir, V. Vijayakumar, Era of Cloud Computing: A New Insight to Hybrid Cloud.* 2015. 51 с.

15. Olowu, M., Yinka-Banjo, C., Misra, S., Florez, H. (2019). A Secured Private-Cloud Computing System. In: Florez, H., Leon, M., Diaz-Nafria, J., Belli, S. (eds) Applied Informatics. . URL: https://doi.org/10.1007/978-3-030-32475-9_27 (дата звернення 20.12.2023).

16. S. Sakr, A. Liu, D. M. Batista and M. Alomari, «A Survey of Large Scale Data Management Approaches in Cloud Environments,» in IEEE Communications Surveys . URL: https://doi.org/10.1109/SURV.2011.032211.00087 (дата звернення 20.12.2023).

17. Kumar Rahul, Rohitash Kumar Banyal, Data Life Cycle Management in Big Data Analytics, Procedia Computer Science, Volume 173. 2020. 371 с.

18. S. Gokulakrishnan and J. M. Gnanasekar, «Data Integrity and Recovery Management in Cloud Systems,» 2020 Fourth International Conference on Inventive Systems and Control (ICISC), Coimbatore, India, 2020. 648 с.

19. W. A. Jansen, «Cloud Hooks: Security and Privacy Issues in Cloud Computing,» 2011 44th Hawaii International Conference on System Sciences, Kauai, HI, USA, 2011, 510с.

20. Ganong, Ray. «The emergence of e-vaulting: electronic vaulting is a compelling improvement on traditional in-house data backup and recovery functions.» Information Management Journal. URL: https://link.gale.com/apps/doc/A97393127/AONE?u=anon~d7b15706&sid=googleScholar&xid=9d845e Accessed 16 Dec. 2023 (дата звернення 20.12.2023).

21. Swagatika S, Niranjan P. Cloud-based backup and data recovery, Journal of Information and Optimization Sciences, 2020. 932 с.

22. Y. Won, R. Kim, J. Ban, J. Hur, S. Oh and J. Lee, «PRUN : Eliminating Information Redundancy for Large Scale Data Backup System,» 2008 International Conference on Computational Sciences and Its Applications, Perugia, Italy, 2008. 144с.

23. Darren Quick, Kim-Kwang Raymond Choo. Google Drive: Forensic analysis of data remnants: Journal of Network and Computer Applications,Volume 40, 2014 193с.

24. *Bulbul Gupta,Pooja Mittal. A Review on Amazon Web Service (AWS), Microsoft Azure & Google Cloud Platform (GCP) Services.* 2021. 504 с.

25. C. Kotas, T. Naughton and N. Imam, «A comparison of Amazon Web Services and Microsoft Azure cloud platforms for high performance computing,» 2018. 157с.

Лістинги програми

Лістинг 1. Код класу «ReviewFileForm»

```
using RepositoryDocumentsApp.Providers;
using System;
using System.Windows.Forms;
namespace RepositoryDocumentsApp.Forms.Controls {
    // Визначення класу ReviewFileForm, який є формою для перегляду файлів
    public partial class ReviewFileForm : Form {
        // Ініціалізація провайдера для взаємодії з файлами Drive
        DriveFileProvider _DriveFileProvider = new DriveFileProvider();
        // Список для зберігання даних про файли з Drive
        List<DriveFile> _DriveFileList = new List<DriveFile>();

        // Конструктор класу ReviewFileForm
        public ReviewFileForm() {
            InitializeComponent(); // Ініціалізація компонентів форми
            // Отримання всіх файлів Drive і збереження їх у _DriveFileList
            _DriveFileList = _DriveFileProvider.GetAllDriveFiles();

            // Заповнення TreeView даними про файли
            PopulateTreeView(0, null);
        }

        // Метод для заповнення TreeView файлами
        private void PopulateTreeView(int level, TreeNode parentNode) {
            // Фільтрація файлів за рівнем вкладеності
            List<DriveFile> childFiles = _DriveFileList.FindAll(file => file.Level == level);
```

```
        foreach (DriveFile file in childFiles) {
            // Створення нового вузла TreeView для кожного файлу
            TreeNode newNode = new TreeNode(file.Name) { Tag = file.Id };
            if (parentNode == null) {
                // Додавання вузла в корінь TreeView, якщо parentNode є null
                FileListTreeView.Nodes.Add(newNode);
            } else {
                // Додавання вузла як дочірнього, якщо parentNode не є null
                parentNode.Nodes.Add(newNode);
            }
            // Рекурсивний виклик методу для папок
            if (file.MimeType == "application/vnd.google-apps.folder") {
                PopulateTreeView(level + 1, newNode);
            }
        }
    }
  }
}
```

Лістинг 2. Код класу «UploadFileForm»

```
using RepositoryDocumentsApp.Providers;
using System;
using System.Collections.Generic;
using System.ComponentModel;
using System.Windows.Forms;
using static System.Windows.Forms.VisualStyles.VisualStyleElement;

namespace RepositoryDocumentsApp.Forms.Controls {
  public partial class ReviewFileForm : Form {
    // Створюємо посилання на провайдера для отримання файлів
```

```
DriveFileProvider _DriveFileProvider = new DriveFileProvider();

// Створюємо список для зберігання файлів
List<DriveFile> _DriveFileList = new List<DriveFile>();

public ReviewFileForm() {
  InitializeComponent();

  // Отримуємо список файлів від провайдера
  _DriveFileList = _DriveFileProvider.GetAllDriveFiles();

  // Викликаємо метод для створення дерева файлів у вікні
  PopulateTreeView(0, null);
}
// Метод для рекурсивного створення дерева файлів у вікні
private void PopulateTreeView(int level, TreeNode parentNode) {
  // Отримуємо список файлів на поточному рівні
  List<DriveFile> childFiles = _DriveFileList.FindAll(file => file.Level == level);
  foreach (DriveFile file in childFiles) {
    // Створюємо новий вузол для відображення файлу
    TreeNode newNode = new TreeNode(file.Name);
    newNode.Tag = file.Id;
    // Додаємо вузол до дерева, враховуючи, чи є батьківський вузол
    if (parentNode == null) {
      FileListTreeView.Nodes.Add(newNode);
    } else {
      parentNode.Nodes.Add(newNode);
    }
```

```
        // Якщо це папка, рекурсивно викликаємо PopulateTreeView для створення піддерева
        if (file.MimeType == "application/vnd.google-apps.folder") {
          PopulateTreeView(level + 1, newNode);
        }
      }
    }
  }
}
```

Скрипти створення бази даних

```
CREATE DATABASE [ELDOC]
CONTAINMENT = NONE
ON PRIMARY
(   NAME = N'ELDOC',
   FILENAME = N'C:\Program Files\ELDOC.mdf',
   SIZE = 5120KB,
   MAXSIZE = UNLIMITED,
   FILEGROWTH = 1024KB
)
LOG ON
(   NAME = N'ELDOC_log',
   FILENAME = N'C:\Program Files\ELDOC.mdf',
   SIZE = 1024KB,
   MAXSIZE = 2048GB,
   FILEGROWTH = 10%
)
GO
USE [ELDOC]
GO
CREATE TABLE [dbo].[DriveFiles](
   [DriveFileId] [int] IDENTITY(1,1) NOT NULL,
   [Id] [char](500) NOT NULL,
   [Name] [char](max) NOT NULL,
   [MimeType] [char](max) NOT NULL,
   [Level] [int] NOT NULL,
   PRIMARY KEY CLUSTERED ([DriveFileId] ASC)
) ON [PRIMARY] TEXTIMAGE_ON [PRIMARY]
```

```
GO
CREATE TABLE [dbo].[Logs](
   [LogsId] [int] IDENTITY(1,1) NOT NULL,
   [UsersId] [int] NOT NULL,
   [EventNameShow] [char](max) NOT NULL,
   [EventDate] [datetime] NOT NULL,
   PRIMARY KEY CLUSTERED ([LogsId] ASC)
) ON [PRIMARY] TEXTIMAGE_ON [PRIMARY]
GO
CREATE TABLE [dbo].[OptionsG](
   [OptionsGId] [int] IDENTITY(1,1) NOT NULL,
   [FilePath] [char](max) NOT NULL,
   [ApplicationName] [char](500) NOT NULL,
   PRIMARY KEY CLUSTERED ([OptionsGId] ASC)
) ON [PRIMARY] TEXTIMAGE_ON [PRIMARY]
GO
CREATE TABLE [dbo].[Users](
   [UsersId] [int] IDENTITY(1,1) NOT NULL,
   [FirstName] [char](50) NOT NULL,
   [LastName] [char](50) NOT NULL,
   [UsersName] [char](50) NOT NULL,
   [UsersPassword] [char](50) NOT NULL,
   [RoleId] [int] NOT NULL,
   [Email] [char](150) NOT NULL,
   [Description] [char](1000) NOT NULL,
   PRIMARY KEY CLUSTERED ([UsersId] ASC)
) ON [PRIMARY]
GO
ALTER DATABASE [ELDOC] SET  READ_WRITE
GO
```